1984

Digital processing of compressed image data

Andrew Masia

DIGITAL PROCESSING OF COMPRESSED IMAGE DATA

by

Andrew Masia

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in the School of
Photographic Arts and Sciences in the
College of Graphic Arts and Photography
of the Rochester Institute of Technology

February, 1984

Signature of the Author Name Illegible

Imaging and Photographic
Science Department

Accepted by Ronald Franari

Coordinator, M.S. Degree Program

College of Graphic Arts and Photography
Rochester Institute of Technology
Rochester, New York

CERTIFICATE OF APPROVAL

M.S. DEGREE THESIS

The M.S. Degree Thesis of Andrew Masia
has been examined and approved
by the thesis committee as satisfactory
for the thesis requirement for the
Master of Science degree

Robert Gonsalves
Dr. Robert Gonsalves, Thesis Advisor

John Carson
Professor John Carson

Dr. Edward Granger
Dr. Edward Granger

2/22/84
Date

THESIS RELEASE PERMISSION FORM

ROCHESTER INSTITUTE OF TECHNOLOGY
COLLEGE OF GRAPHIC ARTS AND PHOTOGRAPHY

Digital Processing of Compressed Image Data

I, Andrew Masia, hereby grant permission to the Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproduction will not be for comercial use or profit.

Digital Processing of Compressed Image Data

I, Andrew Masia, hereby grant permission to the Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproduction will not be for comercial use or profit.

DIGITAL PROCESSING OF COMPRESSED IMAGE DATA

by

Andrew Masia

Submitted to the
Imaging and Photographic Science Department
in partial fulfillment of the requirements
for the Master of Science degree
at the Rochester Institute of Technology

ABSTRACT

Certain image processing functions can be implemented more efficiently when the input data is in compressed form. Such an experimental system has been studied and simulated. The system consists of a one-dimensional Differential Pulse Code Modulation (DPCM) compressor, a one-dimensional non-recursive linear filter, and a one-dimensional DPCM decompressor, applied in that order. The implementation is more efficient because the filter is applied to the data in their compressed form, where fewer bits per pixel are required to represent them. A second, more conventional, system that contains the same functional elements but reverses the order of the filtration and decompression operations has also been implemented for comparison to the experimental one.

The differences (errors) between the signals output from the two systems have been modeled and the models validated through experiments. It has been found that the systems can be made to yield equivalent results if certain parameters are constrained. These constraints do not put undue demands on system design nor do they substantially degrade system performance.

Images produced by the two systems are presented and suggestions for additional work are discussed.

ACKNOWLEDGEMENTS

Sincere thanks are extended to Robert Gonsalves who, as thesis advisor, offered invaluable assistance, particularly at certain key points during the algorithm development phases of this work. Jack Finley of the EIKONIX Corporation originally suggested the direction of this research. Mr. Finley and EIKONIX also made available the Image Digitizers and the Laser Beam Recorder used to prepare the images and generously supplied computer time with which to process them. Mr. Richard Remillard helped to prepare the figures for presentation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

The need to compress digital image data in order to save transmission and/or storage hardware has long been recognized[1] and many schemes have been developed to achieve such compression. At the same time, one of the major advantages of representing an image in digital form is the ease with which various restoration or enhancement techniques may be applied to the data. The problem of image data compression is one of information and coding theory; that of enhancement usually (though not always) relies on the tools of linear systems analysis for a solution.

The field of digital image processing is divided quite nicely into three categories[2] - image compression and coding, image enhancement and restoration, and pattern recognition. Although image processing systems are often called upon to perform tasks related to all three categories, the required algorithms are usually applied independently. Thus, if a system is required both to compress image data for storage and sharpen edges for display, the required procedures for compression are applied to the image, it is decompressed, and finally the edges are sharpened and the image displayed.

The purpose of this research is to consider a system that combines aspects of both compression and enhancement techniques into a single integrated set of image processing

algorithms. An image processing system based on such a set of algorithms, some of which might be affected by special purpose hardware, would be both fast and inexpensive. By enhancing image data in its compressed form, where fewer bit manipulations are required, savings in both storage and computational hardware might be realized. A block diagram of the proposed system, and one of a conventional image processing system, are show in figure 1.1.

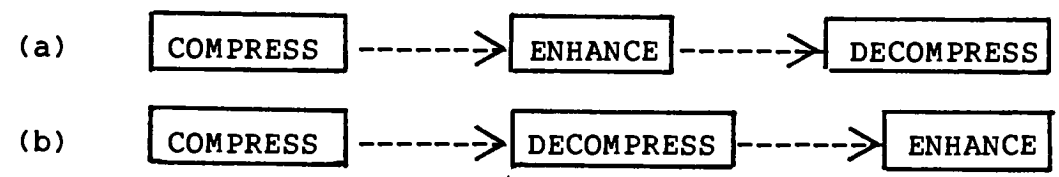

Figure 1.1 Block Diagrams of Experimental (top) and Conventional (bottom) Image Processing Systems

The only difference between the two systems shown in the figure is the order in which the decompression and enhancement operations are applied to the digital data. In the case of the conventional system, (b) of Figure 1.1, the enhancement operator is applied after the decompression so that the required computations need to be done at the higher bit rate. In the case of the proposed system, however, the enhancement operator is applied before decompression so that computations can be made at lower bit rates.

The problem to be investigated here involves the determination of combinations of compression and enhancement procedures that commute, to apply experimentally such procedures in both orders, and to compare the results for

errors and artifacts. It is not the intent of this paper to design or build the hardware that would be required to apply the algorithms in a cost-effective system, but rather to develop and test the computational procedures with software simulations.

2. BACKGROUND

2.1 Image Compression

The simplest method of coding a continuous image, and the one most usually applied, is by sampling the input image function into pixels (picture elements) separated by distances Δx and Δy in the x and y directions respectively and by quantizing the value at each pixel location with a quantizer of uniform step size. If the quantizer output consists of $N = 2^n$ possible levels, then it is a simple matter to encode these values with an n bit natural code by assigning a string of n binary digits representing the appropriate value to be associated with each sample (pixel). This method is often referred to as pulse code modulation[3,4] (PCM).

The advantages of PCM include its ease of implementation with readily available electro-optical and electronic devices, its simplicity and predictability due to the uniform step size of the quantizer, and the fact that equal code length words are produced for each pixel. The latter are compatible with existing general purpose computing machinery. The disadvantage of PCM is that it is inefficient: it uses more bits per pixel than are necessary to represent the image.

With PCM the number of bits required to code the image

is equal to the product of the number of bits to code a pixel, n, and the number of pixels in the image. For example, typically a digital image might consist of 1024 rows and 1024 columns of pixels and be quantized with n = 8 (256 different levels of gray) so that the required number of bits is 8,388,608, or 8 Megabits (1 Megabyte). Shannon has shown[5] that if the statistics of the signal are taken into account in the design of the encoder, it is possible to code the signal with as few as H bits per sample; where H is the entropy of the signal.

As an information source a digital image can be modeled as an M'th order Markov source. As such its entropy, H, is given by

$$H = \lim_{M\to\infty} F_M = - \lim_{M\to\infty} \sum_{i,j} p(B_i,S_j)\log(S_j|\ B_i). \qquad 2.1.$$

In equation 2.1 the S_j are all the possible values that a pixel can assume. The B_i are all the possible states of the surrounding M pixels. If the logarithm is taken to the base two (as is customary) then the units of H are bits per pixel (BPP). The sum in equation 2.1 is taken over all possible combinations (j,i) of gray values and states. For an 8-bit digital image, calculation even of an estimate of H requires summing 256^{M+1} terms. If the pixel value at position (k,j) is correlated with its nearest neighbors only two deep in each direction (vertically, horizontally, and diagonally) then M = 24 and there are 1.6×10^{60} terms in the sum.

Calculation of equation 2.1 is a difficult task even if the required conditional probabilities are known. Furthermore, obtaining reliable estimates of these probabilities is impossible unless further assumptions are made about the image statistics.

As a numerical example of the meaning of entropy as it relates to a digital image, consider two 8-bit PCM digital pictures. The first has a uniform probability density function (pdf) or histogram, all 256 possible gray levels (numbered 0 through 255) being equally likely. The second picture consists only of two different values, half the data being at one level and half at the other level. Assume for the moment that the gray level at any point in either of the pictures is completely independent of the values at the surrounding locations. In this case the source is referred to as a zero memory one (since the points are not correlated with previous samples) and equation 2.1 can be simplified.

$$H = F_0 = -\sum_j p(S_j) \log[p(S_j)]. \qquad 2.2$$

In the case of the first image, $p(S_j)=1/256$ for all j and, since $\log_2(1/256) = -8$, equation 2.2 gives a value of $-(1/256)(-8)(256) = 8$ bits per pixel (BPP). In the case of the second image, however, $p(S_j) = 0.5$ when S_j is either of the two highly likely gray levels and $p(S_j) = 0$ for all other values. Since $\log_2(0.5) = -1$, equation 2.2 gives a value of

(-0.5)(-1)(2) = 1 bit per pixel. What this means is that the first image indeed contains eight bits of information for each of its pixels and 8 BPP are required to transmit the picture with no errors. The second image however contains only one bit per pixel of information and it could be coded for transmission at this lower rate.

It has been shown[6] that for all M, $H \leq F_M$ so that, although it may be impossible to calculate H itself, it is possible to place an upper bound on the image entropy by using a low order model (small M) of the source.

The low source entropy of the digital image stems from two factors: (1) the 256 possible gray levels are not equally probable so the zero memory entropy is not at its maximum of 8 BPP, and (2) the gray level at pixel location (k,j) is highly correlated with those of the surrounding region so that each pixel carries less information than would be expected. (This must be since $H \leq F_M$ for all M.)

It is possible to exploit the first factor simply by recoding the pixel values with an "efficient" code. Methods for deriving such codes from the zero'th order statistics of the image are known.[7,8,9] The greater savings, however, result from the second factor above, and some means of processing the data to decorrelate them is required to reduce the spatial redundancy.

Schemes that have been devised to exploit the high interpixel correlations present in digital images in order to save storage hardware or transmission bandwidth can be

divided into two types: predictive methods and transform methods.[10] Occasionally the two are combined into hybrid systems.[11]

Transform coding methods involve the computation of a non-singular (and often linear) image transform and coding the resulting coefficients with an efficient code. The transform may be one of many including the discrete Fourier (DFT), the discrete cosine (DCT), Hadamard, Harr, or Karhunen-Loeve (KLT) transforms. The method relies on the fact that the high interpixel correlations yield transforms that compress most of the image energy (and thus most of the information) into low sequencies (frequencies only in the case of the DFT). Long codewords are assigned to precisely code the low sequency coefficients that have larger variances, and short codewords are assigned to those of high sequency; hence an efficient code. Since the transform is unitary it is a simple matter to decompress the pictures by decoding the transformed signal and computing the inverse transform.

Of the transforms listed above it can be shown[12] that the KLT is the optimum image transform for energy compaction, but it also is the only one that lacks a fast computational algorithm. For a two-dimensional signal that is autoregressive (interpixel correlations decay exponentially with distance) such as a digital image[13,14] it can also be shown that the KLT basis vectors are closely approximated by those of the DCT.[15] Therefore the two transforms yield

nearly identical results. Since the DCT has a "fast" implementation it is more popular than many of the other transforms when the application is to bandwidth compression. The disadvantage of the transform coding methods is that they require computations over entire images (or at least large image sub-blocks). This means that large amounts of core memory are required. Moreover, some artifacts introduced by the coding are not well understood.

The predictive methods of image compression code the difference between the actual pixel value and some prediction of it based on one or more neighboring pixel(s). Thus high information pixels are those that are poorly predicted and require many bits to code them. Well predicted pixels are coded with a short codeword--thus an efficient code. A particularly successful scheme, called differential pulse code modulation (DPCM), includes a feedback loop as part of the predictor so that quantizing errors do not accumulate.[16,17] A block diagram of a DPCM communication system is shown in figure 2.1. In its simplest form the quantizer, Q, of figure 2.1 is a uniform one and the predictors are single element delays. The system operates in a one-dimensional fashion on a line-by-line basis. In general however the linear predictor is a weighted sum of the reconstructable $\hat{z}$'s and a set of weights, a's.

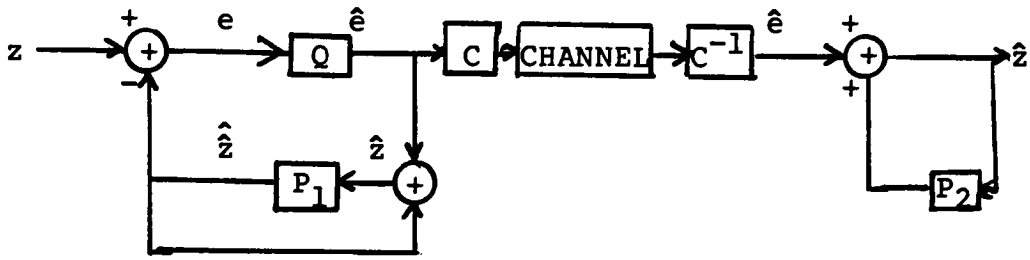

Figure 2.1 DPCM Communication System consisting of Q, Quantizer, P_1, P_2, predictors, C, encoder and C^{-1}, decoder

$$\hat{\hat{z}}(k,j) = \sum_{n>0} \sum_{m>0} a(n,m)\hat{z}(k-n,j-m). \qquad 2.3$$

In equation 2.3, n and m are the distances in the row and column directions respectively that separate the predicted from the predictor pixel. It can be shown[18] that the optimum choices of the a's (at least in terms of a meansquared error statistic) can be computed from the (two-dimensional) autocorrelation function of the digital picture, and that the resulting output signal, ê, is at once zero mean and uncorrelated. It has also been shown[19] that for signals typical of images, near optimum results are obtained when only a few of the a's are nonzero.

Two special cases of DPCM predictors are particularly attractive due to their simplicity and ease of implementation. The first, a single element one-dimensional predictor, results when the a's of equation 2.3 are given by

$$a(n,m) = \begin{cases} 1 & n = 0,\ m = 1 \\ 0 & \text{elsewhere.} \end{cases} \qquad 2.4$$

In this case equation 2.3 reduces to

$$\hat{\hat{z}}(k,j) = \hat{z}(k,j-1). \qquad 2.5$$

The advantage here is that only one storage location is required to affect the prediction, and no computation.

A simple two-dimensional predictor is implemented when the a's are given by

$$a(m,n) = \begin{cases} 0.5 & n=m=1 \\ 0 & \text{elsewhere.} \end{cases} \qquad 2.6$$

In this case equation 2.3 becomes

$$\hat{\hat{z}}(k,j) = 0.5[\hat{z}(k,j-1) + \hat{z}(k-1,j)] , \qquad 2.7$$

and the predicted value is simply the average of the reconstructed pixels directly above and to the left of the pixel being coded. Note that the prediction is always performed using the reconstructed pixel values, since these are available at the decompressor. It is this feature of DPCM that prevents quantizing errors from accumulating. For a raster scanned system, where operations are performed on a row-by-row basis, implementation of DPCM using equation 2.7

for a predictor implies the storage of an entire row of pixels (k-1) as well as one additional pixel (j-1); not an unreasonable requirement.

If equation 2.5 is used as the predictor, and the quantizer of Figure 2.1 is one with uniform step sizes, then the output, $\hat{z}(k,j)$ is identical to a PCM system employing the quantizer, Q, alone.[18] Such a system approximates the input image with only an additive, uncorrelated, noise term. The output samples are given by equation 2.8.

$$\hat{z}(k,j) = z(k,j) + n(k,j). \qquad 2.8$$

In equation 2.8 the n(k,j) are uncorrelated noise samples drawn from a uniformly distributed population with zero mean and variance, $d^2/12$. d is the step size of the quantizer. In the case of the predictor of equation 2.7 the system output is not identical to that of the simple quantizer. However, in this case the statistics of the output image match those of the PCM system.

2.2 Image Restoration Theory

It is well known[20,21,22,23] that many stages of an imaging system are adequately modeled as space-invariant linear systems. Such systems produce degraded versions of input signals that can be described as convolutions of the input with an impulse response, or point spread function (PSF). Due to the convolution-multiplication property of the Fourier transform[24,25] several techniques have been developed

for extracting estimates of the input functions (undegraded images) given the outputs and some prior knowledge of the PSF (or alternately the optical transfer function). These techniques, which are solutions of a convolution integral equation, are known as image restoration procedures and are strictly valid only when the assumptions of linearity and space invariance are also valid. Even when the imaging systems are known to be nonlinear (for instance when they are recorded on photographic film) application of linear filtering operations often yields marked improvements over the degraded pictures. These latter operations will be referred to as image enhancement procedures, although enhancement methods are not limited to linear operations.

The advantage of considering only linear enhancement filters is that they can be easily specified or implemented in either the spatial or the spatial frequency domains, their design considerations have a sound theoretical foundation, and the relationship between their continuous (analog) and discrete (sampled data or digital) representations is well established.

In the continuous spatial domain, a linear filter can be described by the convolution integral

$$g(x,y) = z(x,y) * h(x,y) = \int_{-\infty}^{+\infty}\int z(\nu,\tau)\, h(x-\nu, y-\tau)\, d\nu\, d\tau \quad . \quad 2.9$$

In equation 2.9, g(x,y) is the output of the filter

characterized by its kernel, h(x,y) when its input is z(x,y). In the continuous spatial fequency domain, equation 2.9 is equivalent to

$$G(f_x,f_y) = Z(f_x,f_y) \cdot H(f_x,f_y) \ , \qquad 2.10$$

where $G(f_x,f_y)$ is the Fourier transform of g(x,y) defined by

$$G(f_x,f_y) = \mathcal{F}[g(x,y)] = \int_{-\infty}^{+\infty}\int g(x,y)\exp[-2\pi i(xf_x+yf_y)]dx\ dy \ , \qquad 2.11$$

and Z and H are defined analogously in terms of z and h respectively. In equations 2.10 and 2.11, f_x and f_y are spatial frequency components in the x and y directions and i is the imaginary unit.

In the discrete space of an N-by-N digital image array, the filtering operation is defined in the spatial domain by

$$g(k,j) = \sum_{i=0}^{N-1}\sum_{p=0}^{N-1} z(i,p)h(k-i,j-p) \ . \qquad 2.12$$

Because of the periodic nature of the discrete Fourier transform (DFT), if the indices of equation 2.12 are evaluated modulo(N) the filter may be treated in the spatial frequency domain as

$$G(m,n) = Z(m,n) \cdot H(m,n) \quad . \qquad 2.13$$

In equation 2.13, G(m,n) is the DFT of g(k,j). That is,

$$G(m,n) = \sum_{k=0}^{N-1} \sum_{j=0}^{N-1} g(k,j) \exp[-2\pi i(km+jn)/N]. \qquad 2.14$$

Z and H of equation 2.13 are defined analogously.

The advantage of implementing the filter in the discrete (or digital) domain is that much flexibility exists for changing the type, or parameters of the filters employed (the h's of equation 2.12 or the H's of equation 2.13). Care must be taken in the filter design to prevent the introduction of artifacts that might include ringing, wrap-around effects, overemphasizing of noise, or blurring of edges.

The filter may be implemented by either equation 2.12 or by 2.13. Due to the fast Fourier transform algorithm (FFT) equation 2.13 requires far less computation than 2.12 if many of the h's are nonzero. If all but a few of the filter coefficients are near zero, however, equation 2.12 may be more efficient from a data handling point of view, particularly if data are to be processed "on the fly" or in a pipelined architecture that uses a few rows at a time.

A particularly useful filter, which is optimum in the mean squared error sense when an input image has been

degraded by a linear filter, h, with DFT, H, and corrupted with additive noise, having power spectral density Φ_n, that is independent of the image function, is the Wiener filter,[26] W.

$$W = \frac{H^*}{|H|^2 + \Phi_n / \Phi_z} \qquad 2.15$$

In equation 2.15, Φ_z is the power spectrum of the signal input to the degrading filter H, and the (*) indicates complex conjugation. All of the terms in equation 2.15 are functions of spatial frequency components in both directions. The inverse signal to noise term, Φ_n/Φ_z, is sometimes assumed to be constant and useful results are still obtained.

2.3 Combination of Compression and Restoration Procedures

Since any digital image processing system requires that large amounts of data be manipulated, efficient ways of storing, retrieving, and processing those data are required. This is particularly true if the tasks are to be performed with reasonable amounts of hardware and in near real time (defined arbitrarily as less than a few video frame times).

An obvious method for reducing the storage requirements of the system is to reduce the amount of data to be stored. This can be accomplished with one of the image compression schemes outlined in section 2.1. Unless one is careful, however, reducing the demands on the storage subsystem may

increase the load on the processor. If an image must be decompressed each time it is called up to be processed, and compressed before it is again stored, many additional operations are required to affect the processing since the compression and decompression operations are image processing functions as well. If, on the other hand, the processing can be performed on the data in their compressed state, then the potential savings in storage can be realized with no additional load on the processing system. In fact, the amount of processing required might be reduced since the image itself consists of fewer bits.

In short, an image processing system is sought that compresses the image data on input and requires no surfacing from the compressed domain throughout the processing operations. Only upon output (to either soft or hard copy display devices) should the image be decompressed.

Certainly the design of the entire image processing system is beyond the scope of this research. In its final implementation the compression and decompression, as well as some of the intermediate processing, would need to be affected by special purpose hardware in order to be efficient. The work performed here concerns the development of compression and processing algorithms that are compatible with such a system. These algorithms must be implemented on a general purpose computing machine, and their performance evaluated, to demonstrate the validity and limitations of such a compressed domain digital image processing system.

Likely candidates for enhancement and compression techniques that might lend themselves to combination have been discussed in previous sections. Linear enhancement operations are particularly attractive for several reasons. Their effects are easily analyzed in either the spatial, or the spatial frequency domains. When the filter designs are based on models of the optical image forming process, linear filters can be made to restore images subject to many typical degradations (such as defocus or image motion).

For reasons concerned with data handling, methods that operate on only a few lines of data at a time are attractive. Digital images are often formed by various raster scanning systems and transmitted one line at a time. They are stored on magnetic disks or tapes in a row-sequential fashion. Processing methods that operate on data in this manner would be easily integrated with existing hardware and software systems. If images can be processed as they are transmitted from one device to another (i.e. between microwave receiver and magnetic storage disk or between disk and softcopy display system) then the data need only be handled a minimum number of times. Also, as the data are being transmitted along one of these channels, special purpose hardware can be designed to process them in a pipeline fashion.

It would be advantageous, from an image understanding point of view, if the compression and enhancement operations were commutative; that is, if after reversal of the order of their application, the system output remained unchanged.

This would allow the enhancement operator to be designed using the normal tools of image enhancement, without regard to the fact that it is to be applied to compressed image data.

3. THEORETICAL DEVELOPMENT

3.1 Introduction

In this chapter a model is derived that predicts the difference between an image filtered in the compressed domain of a DPCM communications system and one that is filtered after the image is decompressed. It is shown that except for an additive noise term the systems are identical. An expression for the mean squared error between the outputs of the two systems is developed.

The two systems are shown in figure 3.1. It is first assumed that the step size of the quantizer, Q, in the figure is unity, and that the predictor is a simple, single element delay. These restrictions will be removed later but they greatly simplify the resulting analysis and allow considerable insight to the problem at hand.

A numerical example is given for illustration and intuitive understanding. The results will be derived more formally and generally in later sections.

A portion of a digital image is shown in numerical form in figure 3.2. The number at each location indicates the gray value at the corresponding point.

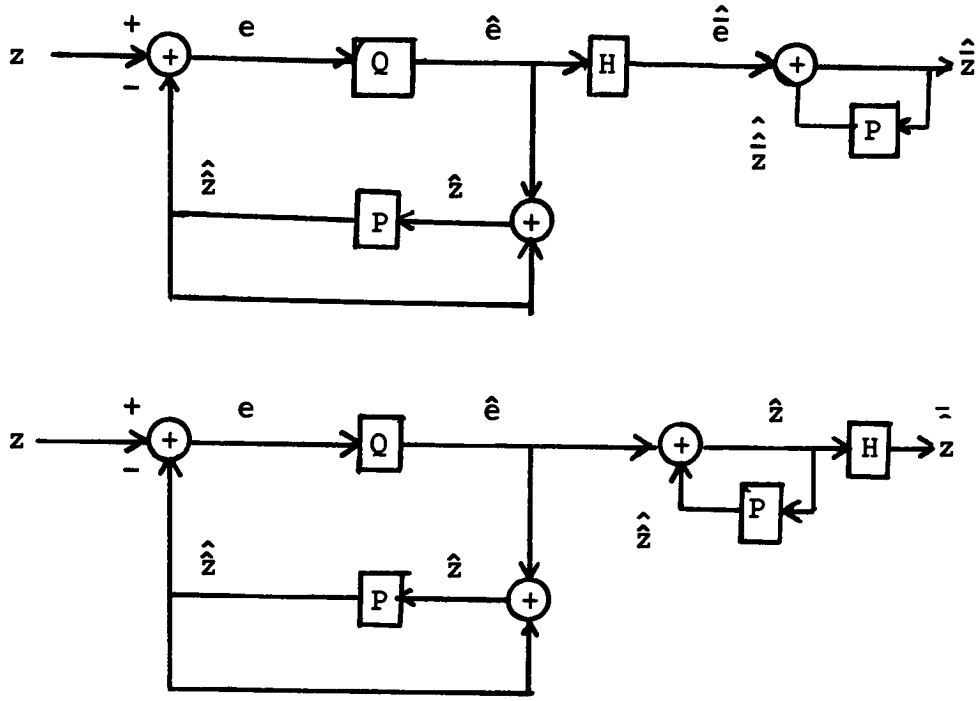

Figure 3.1 Experimental System (top)
and Control System (bottom)

5	8	6	6	10	11
7	2	4	8	9	12
4	4	6	8	10	12
6	8	5	9	5	7

Figure 3.2 Example of an Input Digital Image

Since the predictor is a single element delay across each row of data and the quantizer is of unit step size, it is a simple matter to compute the DPCM error signal, $\hat{e}$, as the

first finite difference of each row. This is shown in figure 3.3.

5	3	-2	0	4	1
7	-5	2	4	1	3
4	0	2	2	2	2
6	2	-3	4	-4	2

Figure 3.3 Example Picture Compressed

In the case of the conventional image processing system the picture is then decompressed. This is done simply by accumulating the sum of all the preceding values in each row of the compressed picture. The result is shown in figure 3.4.

5	8	6	6	10	11
7	2	4	8	9	12
4	4	6	8	10	12
6	8	5	9	5	7

Figure 3.4 Example Picture Decompressed

Note that figure 3.4 is identical to figure 3.2. This is because the quantizer is of unit step size. If it were not, then some quantizing noise would have been added to each of the values of figure 3.4. In the conventional system the decompressed image is then filtered. If the filter used is

that shown in figure 3.5 then, the output of the system is that shown in figure 3.6.

-1	+4	-2

Figure 3.5 Filter Used for Example Pictures

4	15	4	-2	12	...
24	-7	-2	10	4	...
8	0	4	6	8	...
8	16	-6	21	-3	...

Figure 3.6 Example of Output from Conventional Image Processing System

In the case of the experimental image processing system the compressed image shown in figure 3.3 is filtered with the fiter shown in figure 3.5. The result is shown in figure 3.7.

14	11	-11	-6	14	...
38	-31	5	12	-6	...
16	-8	4	2	2	...
20	8	-22	27	-24	...

Figure 3.7 Example Picture Compressed and Filtered

Finally it is a simple matter to decompress the picture shown in figure 3.7 by summing across each row. The final output

of the experimental system is shown in figure 3.8.

14	25	14	8	22	...
38	7	12	24	18	...
16	8	12	14	16	...
20	28	6	33	9	...

Figure 3.8 Example of Output From the Experimental Image Processing System

The error picture can now be defined as the pixel by pixel difference between the output of the two image processing systems shown in figures 3.8 and 3.6. This is shown in figure 3.9.

10	10	10	10	10	...
14	14	14	14	14	...
8	8	8	8	8	...
12	12	12	12	12	...

Figure 3.9
Example of Error Picture Between the Experimental and the Conventional Image Processing Systems

Notice that the error is constant for each row of the image and that it has a value equal to the product of the right-most point of the filter (2) and the first pixel in the row. It will be shown in the following sections that, for an arbitrarily sized filter kernel, the error is the sum of the

products of the right side of the filter with the corresponding first columns of each row. Since these data are available to the image processing system it is a simple matter to compute the required correction terms and apply them to the data on decompression: thus reducing this source of error to zero. It will also be shown however that when the quantizer step size is increased from unity, other sources of error (noise) become apparent.

3.2 Development for Unit Step Sized Quantizer and Predictor

The inputs to the two systems shown in figure 3.1 are identical and consist of the sequence z_j (j = 1,2,3,...N) of non-negative integers. N is the width of the picture, and each row of the picture is considered to be a new sequence of z's. The first operation of the DPCM encoder is to subtract from the input its predicted value, $\hat{z}$, to give the error signal, e.

$$e_j = z_j - \hat{z}_j. \qquad 3.1$$

Since the quantizer is of unit step size it has no effect on the error signal, e.

$$\hat{e}_j = e_j. \qquad 3.2$$

In both of the systems of figure 3.1 it is the quantized error signal, $\hat{e}_j$, that is transmitted through the rest of the system. The signal is added to the current value of the

predictor, $\hat{\hat{z}}$, to form the reconstructed value, $\hat{z}$, which will be used to predict the next pixel value.

$$\hat{z} = e_j + \hat{\hat{z}}. \qquad 3.3$$

Since the predictor in this case is just a single element delay, its output, $\hat{\hat{z}}_j$, is just its previous input.

$$\hat{\hat{z}}_j = \hat{z}_{j-1}. \qquad 3.4$$

Since the quantity $\hat{\hat{z}}_j$ added to e_j to form $\hat{z}_j$ is precisely the same value that was subtracted from z_j to form e_j, and, since equation 3.2 is true under the current restrictions, $\hat{z}_j$ is identical to the input, z_j, for all j. Equation 3.4 can be rewritten:

$$\hat{\hat{z}}_j = z_{j-1}. \qquad 3.5$$

It also follows that equation 3.1 can be rewritten:

$$\hat{e}_j = z_j - z_{j-1}. \qquad 3.6$$

Equation 3.6 can be interpreted as meaning that, under the current restrictions, the output of the DPCM encoder, $\hat{e}$, is just the first finite difference (discrete approximation of the first derivative) of the input signal, z.

In the case of the experimental system of figure 3.1 the next step is to filter the data. The output of a filter with finite support of length M is given by equation 3.7.

$$g_j = \sum_{i=1}^{M} x_{j-i+m}\, h_i, \quad m=(M+1)/2 \quad . \qquad 3.7$$

In equation 3.7, g is the output of the filter with impulse response, h. To ease the notation, M is restricted to be an odd integer so that the filter's midpoint is indexed at m = (M+1)/2. x is the signal input to the filter.

If a filter is applied to the output of the DPCM compressor of equation 3.6 then the output of the second stage of the experimental system, $\hat{\bar{e}}$, is given by equation 3.8.

$$\hat{\bar{e}} = \sum_{i=1}^{M} z_{j-i+m}\, h_i \; - \sum_{i=1}^{M} z_{j-1-i+m}\, h_i \; . \qquad 3.8$$

The final stage of the experimental system is the DPCM decompressor. Just as the compressor (under the current restrictions, soon to be removed) is a simple differentiator, one would expect the decompressor to be a simple integrator. This is indeed the case. For a decompressor input of $\hat{\bar{e}}$, the output is $\hat{\bar{z}}$.

$$\hat{\bar{z}}_j = \hat{\bar{e}}_j + \hat{\bar{z}}_{j-1} \qquad 3.9$$

Equation 3.9 is recursive in $\hat{\bar{z}}_j$ since the last term can be written in terms of the previous pixel. Thus equation 3.9 can be rewritten:

$$\hat{\hat{z}}_j = \sum_{i=1}^{j} \hat{\hat{e}}_i \ . \qquad 3.10$$

By replacing the expression given in equation 3.8 for $\hat{\hat{e}}_j$ into equation 3.10, the experimental system's output can be expressed in terms of the input and the values of the filter kernel.

$$\hat{\hat{z}}_j = \sum_{i=1}^{j} \sum_{k=1}^{M} z_{i-k+m} \, h_k - \sum_{k=1}^{M} z_{i-1-k+m} \, h_k \ . \qquad 3.11$$

Reversing the order of the summation gives

$$\hat{\hat{z}}_j = \sum_{k=1}^{M} h_k \left(\sum_{i=1}^{j} z_{i-k+m} - \sum_{i=1}^{j} z_{i-1-k+m} \right) . \qquad 3.12$$

The second inner sum can be reindexed

$$\hat{\hat{z}}_j = \sum_{k=1}^{M} h_k \left(\sum_{i=1}^{j} z_{i-k+m} - \sum_{i=0}^{j-1} z_{i-k+m} \right) . \qquad 3.13$$

Finally, the last term of the first inner sum and the first term of the second can be stripped off.

$$\hat{\hat{z}}_j = \sum_{k=1}^{M} h_k \left(z_{j-k+m} + \sum_{i=1}^{j-1} z_{i-k+m} - \sum_{i=1}^{j-1} z_{i-k+m} - z_{-k+m} \right) . \quad 3.14$$

Note that the two inner sums cancel and the expression simplifies.

$$\hat{\hat{z}}_j = \sum_{k=1}^{M} h_k \, z_{j-k+m} - \sum_{k=1}^{M} h_k \, z_{-k+m} . \quad 3.15$$

The first term of equation 3.15 is simply the filtered version of the input to the system (the desired result) and the second term is independent of j, the pixel column index. Thus the entire second term is an error term that is applied to the entire row and tha depends only on the first m columns of the picture row and one half of the filter kernel.

In the case of the conventional system in figure 3.1 the decompressor immediately follows the compressor. Its output, $\hat{z}$, is given by equation 3.16.

$$\hat{z}_j = \hat{e}_j + \hat{\hat{z}}_j . \quad 3.16$$

Since the predictor is a simple one-element delay, equation 3.16 can be rewritten.

$$\hat{z}_j = \hat{e}_j + \hat{z}_{j-1} \cdot \qquad 3.17$$

Equation 3.17 is recursive in $\hat{z}$ since each value depends on the previous one.

$$\hat{z}_j = \sum_{i=1}^{j} \hat{e}_i \cdot \qquad 3.18$$

The expression for $\hat{e}_j$ derived in equation 3.6 can be replaced into equation 3.18.

$$\hat{z}_j = \sum_{i=1}^{j} (z_i - z_{i-1}) \cdot \qquad 3.19$$

Equation 3.19 can be simplified by splitting the right side into the difference of two sums, changing the index of the second sum, removing the last term from the first sum and the first term from the last, and cancelling appropriate terms.

$$\hat{z}_j = \sum_{i=1}^{j} z_i - \sum_{i=1}^{j} z_{i-1} \cdot \qquad 3.20$$

$$\hat{z}_j = \sum_{i=1}^{j} z_i - \sum_{i=0}^{j-1} z_i \cdot \qquad 3.21$$

$$\hat{z}_j = z_j + \sum_{i=1}^{j} z_i - \sum_{i=1}^{j} z_i - z_0 . \qquad 3.22$$

$$\hat{z}_j = z_j - z_0 . \qquad 3.23$$

Recall that the signal, z_j, is defined only for $j \geq 1$. However, z_0 can be defined to be equal to zero. This is equivalent to starting the DPCM compressor with zeros loaded into all of the memory locations, a simple task in both the software simulations and in any hardware built to perform these tasks. Initializing all of these values to zero gives a prediction for the first pixel of zero. Thus that pixel is transmitted across the channel unaltered. Under the present restrictions (unit quantizer step size and single element delay predictor) equation 3.23 can be rewritten:

$$\hat{z}_j = z_j . \qquad 3.24$$

The final stage of the control image processing system is to filter the output of the decompressor. Thus, the output of the conventional system, $\bar{\hat{z}}$, is

$$\bar{\hat{z}}_j = \sum_{k=1}^{M} z_{j-k+m} h_k . \qquad 3.25$$

Note that $\bar{\hat{z}}_j$ is identical to the first term of the expression

for $\hat{\bar{z}}_j$ derived in equation 3.15. An error function, ϵ_j, can be defined as the difference between the experimental output value and that of the conventional system on a pixel-by-pixel basis. In this case

$$\epsilon_j = \hat{\bar{z}}_j - \bar{\hat{z}}_j. \qquad 3.26$$

Substituting the appropriate expressions into 3.26 and simplifying, we get

$$\epsilon_j = -\sum_{k=1}^{M} h_k z_{-k+m}. \qquad 3.27$$

Note that the error is independent of the column index, j. It is a constant for the entire row and requires only the first M pixels of the picture and the filter kernel for its computation. The error term can easily be computed at the input to the compressor and transmitted at the beginning of each line. Little additional overhead is required.

3.3 Development for Arbitrary Quantizer Step Size and Predictor

In the previous development several simplifying assumptions were made. It was assumed that the step size of the quantizer was unit and that the predictor, P in figure 3.1, was a simple one-element delay. Also, the statement of equation 3.7 carries with it the implicit assumption that the computation of the filter equation introduces no noise. If

the filter is to be implemented by a finite state computing machine, then some round-off error is inevitable. In fact, if the output of the filter is to take on only integer values, then any computation ends with a quantizer of step size d. In a conventional image processing system these quantizing effects are usually small compared to the noise in the system and they can be ignored. In the case of the compressed domain digital image processing system however, noise added to the low redundancy signals outside of the compressor's feedback loop can be disastrous.

With the assumption of a noise-free computation of the filter removed, equation 3.7 can be rewritten to include the effect of the quantizer:

$$g_j = \sum_{i=1}^{M} x_{j-i+m}\, h_i \quad + n_j \ . \qquad 3.28$$

In equation 3.28, g is the output of the filter with impulse response h. M is the size of the filter (assumed to be an odd integer for ease of notation) and m=(M+1)/2 is the index of the midpoint of the filter's impulse response. x is the signal input to the filter and the n_j are independent samples drawn from a uniformly distributed random population of width d and mean zero. The variance, σ^2 of such a population is $d^2/12$.

Reference is again made to figure 3.1. It will be

assumed that data within the DPCM compression and decompression loops can be maintained at full precision but that the input to the system, the output of the compressor, $\hat{e}$, and that of the filter, $\hat{\bar{e}}$ in the case of the experimental system and $\hat{\bar{z}}$ in the case of the control system can take on only integer values. The predictor, P, operates only on the previous input but is capable of scaling as well as delay.

Algebraically the uniform step size quantizer with step size d is represented by equation 3.29.

$$Q\{x_i\} = \hat{x}_i = x_i + n_i \; . \qquad 3.29$$

In equation 3.28, Q is the quantizing operator, x_i is the i'th input to Q, $\hat{x}_i$ is the i'th output and n_i is the i'th sample of noise drawn from a uniformly distributed independent population of mean 0 and variance $d^2/12$.

The predictor output is given by equation 3.30 when its input is x_i.

$$P\{x_i\} = \hat{x}_i = ax_{i-1} \; . \qquad 3.30$$

Equation 3.30 shows the predictor output as the product of a constant, a, with the previous sample input.

Examination of figure 3.1 indicates that in both systems the output of the compressor, $\hat{e}$, is as shown in equation 3.31.

$$\hat{e}_j = z_j - az_{j-1} - an_{j-1} + n_j \; . \qquad 3.31$$

In the experimental system the DPCM output is then filtered and quantized according to equation 3.28.

$$\hat{\hat{e}}_j = \sum_{i=1}^{M} \hat{e}_{j-i+m}\, h_i + n_j \qquad 3.32$$

Finally the compressed and filtered signal, $\hat{\hat{e}}$, is decompressed. Since the predictor used in the decompressor is the same as the one used in the compressor, its output, $\hat{\hat{z}}_j$, is just the sum of the current input and the past output weighted by the factor a.

$$\hat{\hat{z}}_j = \hat{\hat{e}}_j + a\hat{\hat{z}}_{j-1} \; . \qquad 3.33$$

If $0<a<1$ then equation 3.33 represents a "leaky" integrator. This expression for the system output represents a recursive filter since $\hat{\hat{z}}_{j-1}$ can be expressed in terms of $\hat{\hat{z}}_{j-2}$ and $\hat{\hat{e}}_{j-1}$. In fact equation 3.33 can be used as a recurrence relationship to derive equation 3.34.

$$\hat{\hat{z}}_j = \sum_{i=1}^{j} a^{j-i}\, \hat{\hat{e}}_i \; . \qquad 3.34$$

It is now possible to develop an expression for the system output directly in terms of its input, the filter impulse response, and the appropriate noise samples.

First the right side of equation 3.32 is substituted into equation 3.34 for $\hat{\bar{e}}_i$.

$$\hat{\bar{z}}_j = \sum_{i=1}^{j} a^{j-i} \sum_{k=1}^{M} \hat{\bar{e}}_{i-k+m}\, h_k + n_i \,. \qquad 3.35$$

Since the noise sample is independent of the inner (filter) sum the terms can be regrouped.

$$\hat{\bar{z}}_j = \sum_{i=1}^{j} a^{j-i} \sum_{k=1}^{M} \hat{e}_{i-k-m}\, h_k + \sum_{i=1}^{j} a^{j-i}\, n_i \,. \qquad 3.36$$

Next, equation 3.31 is substituted into equation 3.36 for $\hat{e}_{i-k+m}$.

$$\hat{\bar{z}}_j = \sum_{i=1}^{j} a^{j-i} \sum_{k=1}^{M} (z_{i-k+m} + n_{i-k+m} - z_{i-k+m-1} - n_{i-k+m-1}) h_k + \sum_{i=1}^{j} a^{j-i}\, n_i \,. \qquad 3.37$$

The first set of sums can be broken into two. This is shown in equation 3.38.

$$\hat{\tilde{z}}_j = \sum_{i=1}^{j} a^{j-i} \sum_{k=1}^{M} h_k (z_{i-k+m} + n_{i-k+m})$$

$$- \sum_{k=1}^{M} h_k (z_{i-k+m-1} + n_{i-k-m-1})$$

$$+ \sum_{i=1}^{j} a^{j-i} n_i \, . \qquad 3.38$$

The factor of a^{j-i} can then be distributed over the pair of inner sums.

$$\hat{\tilde{z}}_j = \sum_{i=1}^{j} \sum_{k=1}^{M} a^{j-i} h_k (z_{i-k+m} + n_{i-k+m})$$

$$- \sum_{k=1}^{M} a^{j-i+1} h_k (z_{i-k+m-1} + n_{i-k+m-1})$$

$$+ \sum_{i=1}^{j} a^{j-i} n_i \, . \qquad 3.39$$

By removing the last term of the first outer sum and the first term of the second outer sum, we can rewrite equation 3.39.

$$\hat{\hat{z}}_j = \sum_{i=1}^{j-1} \sum_{k=1}^{M} a^{j-i} \, h_k (z_{i-k+m} + n_{i-k+m})$$

$$- \sum_{i=2}^{j} \sum_{k=1}^{M} a^{j-i+1} \, h_k (z_{i-k+m-1} + n_{i-k+m-1})$$

$$+ \sum_{k=1}^{M} a^{j-i} \, h_k (z_{j-k+m} + n_{j-k+m})$$

$$- \sum_{k=1}^{M} a^{j-1+1} (z_{1-k+m-1} + n_{1-k+m-1})$$

$$+ \sum_{i=1}^{j} a^{j-1} \, n_i \, . \qquad 3.40$$

Next, a substitution is made in the second double sum. Let the index of the sum be $\hat{i} = i-1$. Then $i = \hat{i}+1$. When $i = 2$, the starting index of the sum, $\hat{i} = 1$; when $i = j$, the terminal index of the sum, $\hat{i} = j-1$. The exponent on a becomes $j-(\hat{i}+1)+1 = j-\hat{i}$ and the subscripts on the z and n terms can be rearranged. $(i-1)-k+m = \hat{i}-k+m$. Substituting these results into equation 3.40 we get equation 3.41.

$$\hat{\tilde{z}} = \sum_{i=1}^{j-1} \sum_{k=1}^{M} a^{j-i} h_k(z_{i-k+m} + n_{i-k+m})$$

$$- \sum_{\hat{i}=1}^{j-1} \sum_{k=1}^{M} a^{j-\hat{i}} h_k(z_{\hat{i}-k+m} + n_{\hat{i}-k+m})$$

$$+ \sum_{k=1}^{M} h_k(z_{j-k+m} + n_{j-k+m})$$

$$- \sum_{k=1}^{M} a^{j}(z_{-k+m} + n_{-k+m})$$

$$+ \sum_{i=1}^{j} a^{j-i} n_i \quad . \qquad 3.41$$

Since the new index, $\hat{i}$, is a dummy variable, the second double sum is identical to the first, except for its sign, and the two sum to zero. Equation 3.41 can now be simplified by removing the vanishing terms and factors from the remaining sums and regrouping.

$$\hat{\tilde{z}}_j = \sum_{k=1}^{M} h_k(z_{j-k+m} + n_{j-k+m}) - a^j \sum_{k=1}^{M} h_k(z_{-k+m} + n_{-k+m}) + \sum_{i=1}^{j} a^{j-i} n_i \; . \qquad 3.42$$

The first sum in equation 3.42 is the "desired" result, the filtered version of the system input plus the noise introduced by the compression and decompression processes. The second term is similar to the error term derived in equation 3.27 but includes the effects of the compressor quantizing noise. The sum is independent of the column index, j, and depends only on the first M pixels of the input row and the filter coefficients. In this case, however, the second sum is scaled by the factor a^j. For a even slightly less than unity this factor decreases rapidly with j, thus dampening out the effects of the error term across the width of the picture. The third term represents the effects of the noise added by the filter. Note that there is a contribution to this term not only from the j'th sample value, but from all the preceding samples in the row. This is because the noise is introduced outside the feedback loop of the compressor.

In the case of the conventional system of figure 3.1 the output of the compressor as shown in equation 3.31 is immediately input to the decompressor, which has output given by equation 3.43.

$$\hat{z}_j = \sum_{i=1}^{j} a^{j-i} \hat{e}_i \ . \qquad 3.43$$

Replacing the right side of equation 3.30 for $\hat{e}_i$ in equation 3.43 and regrouping terms, we get

$$\hat{z}_j = \sum_{i=1}^{j} a^{j-i}(z_i+n_i) - a \sum_{i=1}^{j} a^{j-i}(z_{i-1}+n_{i-1}) \ . \qquad 3.44$$

By removing the last term from the first sum, reindexing the second sum, and removing its first term equation, we get

$$\hat{z}_j = z_j+n_j + \sum_{i=1}^{j-1} a^{j-i}(z_i+n_i) - \sum_{i=1}^{j-1} a^{j-i}(z_i+n_i) - a_j(z_0+n_0) \ . \qquad 3.45$$

In equation 3.45 the two summations sum to zero and the last term is zero since the index ,0, is outside the domain of the picture. Thus in the case of the conventional system the

output of the decompressor is just the sum of the system input and a single noise term,

$$\hat{z}_j = z_j + n_j \quad . \qquad 3.46$$

The final stage of the control system is to filter $\hat{z}$, the output of the decompressor. Applying equation 3.28 to equation 3.46, we get,

$$\bar{\hat{z}}_j = \sum_{k=1}^{M} (z_{j-k+m} + n_{j-k+m}) h_k \quad + \quad n_j \quad . \qquad 3.47$$

Equation 3.47 shows the output of the control system as the filtered image plus compressor noise with an additional single noise sample due to the integer output of the filter.

Having derived expressions for the outputs of the two image processing systems in equations 3.42 and 3.47, we can now define the difference, or error, between them. Let ϵ_j be the difference between corresponding points in the experimental and the control systems.

$$\epsilon_j = \hat{\bar{z}}_j - \bar{\hat{z}}_j \quad . \qquad 3.48$$

Equation 3.49 is derived by substituting equations 3.42 and 3.47 into equation 3.48.

$$\epsilon_j = - a^j \sum_{k=1}^{M} h_k (z_{-k+m} + n_{-k+m}) + \sum_{k=1}^{M} a^{j-i} n_i \quad - n_j \quad . \quad 3.49$$

The value of the first sum in equation 3.49 depends only on the first few columns of the picture and the filter kernel. It is a simple matter to compute this sum of each row of the picture and correct the final output image on decompression. This can be accomplished simply by initializing the delay buffer of the decompressor with the first sum of equation 3.49. If this is done the noise term at position j across the row, N_j, is given by equation 3.50.

$$N_j = \sum_{i=1}^{j} a^{j-i} n_i \quad - \quad n_j \quad . \quad 3.50$$

Recall that the n_i are independent samples drawn from a uniformly distributed population of mean 0 and variance $d^2/12$. It is thus a simple matter to calculate the mean and variance of the N_j. The mean is given by equation 3.51.

$$\mu(N_j) = \sum_{i=1}^{j} a^{j-i} \mu(n_i) - \mu(n_j) = 0 \quad . \quad 3.51$$

This is true since the mean of a sum of random variables,

each weighted by a different coefficient, is simply the sum of the means of the individual random variables, each weighted by the corresponding coefficient.[27] In this case $\mu(n_i) = 0$ for all i. Similarly, the variance of the random variable N_j is just the sum of the variances of the n_i, each weighted by the square of the corresponding coefficient[28] since the n_i are uncorrelated.

$$\sigma^2(N_j) = \sum_{i=1}^{j} a^{2(j-i)} \sigma^2(n_i) + \sigma^2(n_j) . \qquad 3.52$$

In equation 3.52 the factor of a^{2j} can be removed from the sum, as can $d^2/12$, the variance of all the n_i.

$$\sigma^2(N_j) = a^{2j}(d^2/12) \sum_{i=1}^{j} a^{-2i} + (d^2/12) . \qquad 3.53$$

When a=1, that is the predictor is a simple one-element delay, equation 3.53 reduces to

$$\sigma^2(N_j) = (d^2/12) \sum_{i=1}^{j} 1 + (d^2/12)$$

$$= (j+1)(d^2/12) , \quad \text{for } a=1 . \qquad 3.54$$

However, when 0<a<1, equation 3.53 can be rewritten:

$$\sigma^2(N_j) = a^{2j}(d^2/12) \sum_{i=1}^{j} (a^{-2})^i + (d^2/12) . \qquad 3.55$$

Now, since 0<a<1, 1/a > 1. The sum in equation 3.55 is simply a geometric series with ratio $1/a^2$ greater than one. Such a series is divergent in the limit as j goes to infinity, but the sum in equation 3.55 has a finite upper limit. The sum of k terms of a geometric series with ratio r and common factor b is given in equation 3.56.[29]

$$\sum_{i=0}^{k-1} br^i = b(r^k-1)/(r-1) . \qquad 3.56$$

In the case of the sum of equation 3.55, the ratio is a^{-2} and the common factors have already been removed from the sum. Compensating for the difference in indexing between the sums of equations 3.55 and 3.56, we can eliminate the sum in equation 3.55, which can now be rewritten

$$\sigma^2(N_j) = a^{2j} \frac{d^2}{12} \frac{a^{-2(j+1)}-1}{a^{-2}-1} + \frac{d^2}{12} . \qquad 3.57$$

Equation 3.57 can be further simplified using simple algebraic manipulations.

$$\sigma^{2}(N_j) = \frac{d^2}{12}\left[\frac{1-a^{2(j+1)}}{1-a^2} + 1\right]. \qquad 3.58$$

It is not surprising to note that the mean squared error at position j across the row scales linearly with $d^2/12$, the variance of the individual noise samples. The second factor, however, defines the position dependence as well as the dependence on the parameter, a, and is well worth examining.

Let

$$E(j|a) = \frac{1- a^{2(j+1)}}{1- a^2} + 1 . \qquad 3.59$$

When a equals unity, equation 3.59 is undefined. However, using l'Hôpital's rule it is found that

$$\lim_{a\to 1} E(j|a) = j + 1 . \qquad 3.60$$

Equation 3.60 is just a restatement of equation 3.54 with a slightly different derivation. These expressions indicate that when a=1 the mean square error increases linearly across the rows of the picture. The bigger the picture gets, the noisier will be the right hand side. For 0<a<1 however, the mean squared error becomes asymptotic to some limiting value since, in this case,

$$\lim_{j\to\infty} E(j|a) = \frac{1}{1 - a^2} + 1 . \qquad 3.61$$

The only question that remains is the rate at which the function converges to its limiting value. This is best shown graphically, so the function E(j|a) is plotted in figure 3.10 for several values of a. Note that for a equal to 0.95, 0.9, 0.85, and 0.8 E(j|a) climbs to within 90% of its limiting value within 45, 18, 11, and 7 pixels respectively.

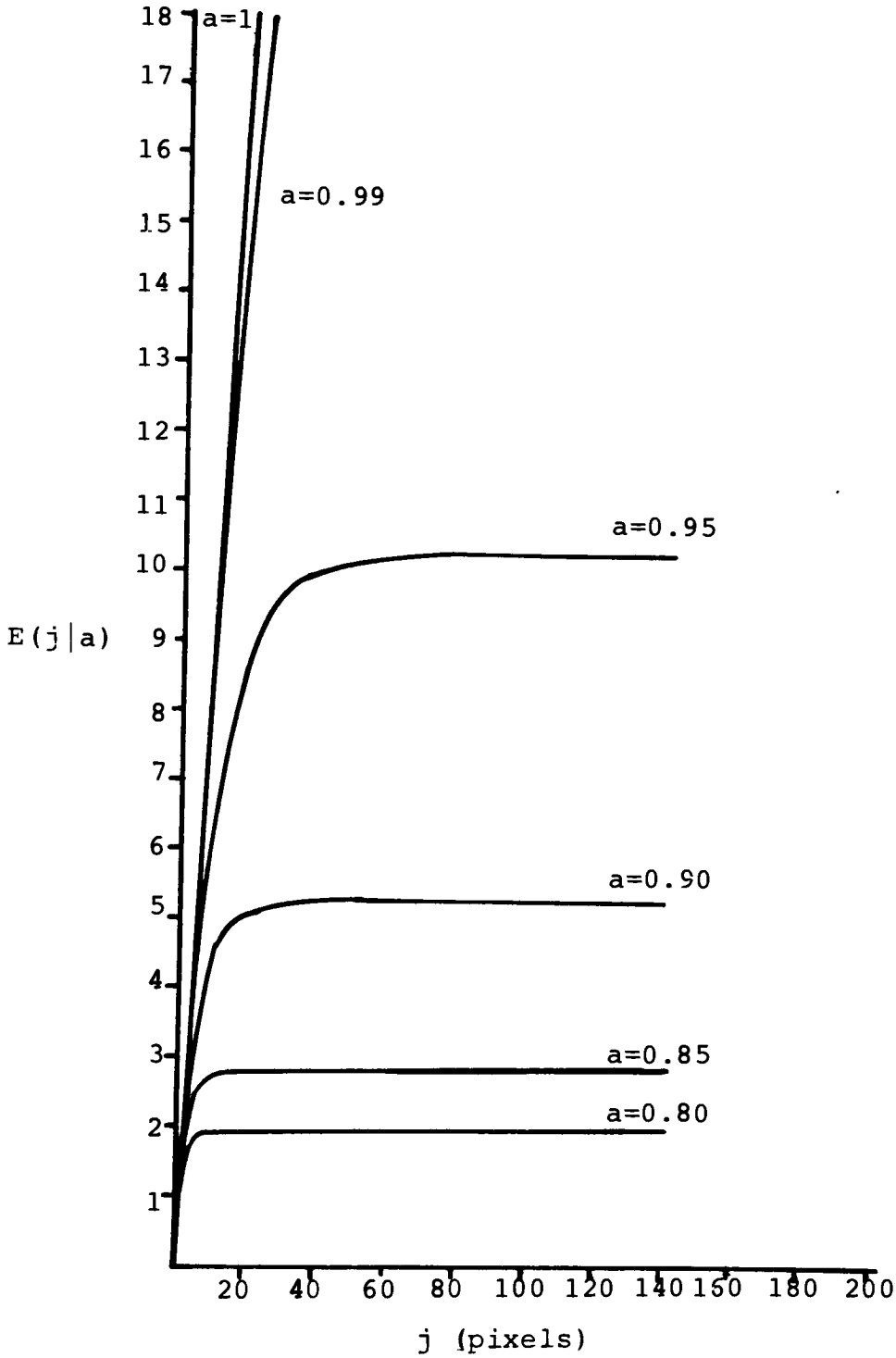

Figure 3.10 Mean Squared Error Between Experimentally and Conventionally Processed Pictures as Functions of the Column Index, j, for Various Values of the Prediction Coefficient, a.

4. EXPERIMENTAL

4.1 Image Processing

Two computer programs were written in the FORTRAN programming language to simulate the systems shown in figures 3.1a and 3.1b. The programs run on a Digital Eqipment Corporation VAX 11/780 computer operating under the VAX/VMS operating system. The primary consideration when writing the code was readability. Because this implementation of the algorithms was meant only as a "proof of concept," little attempt was made to write "efficient" programs, either from program size or execution speed points of view. Thus, no multiple buffering or parallel processing is employed. It is hoped that the avoidance of such programming techniques contributes to the readability of the resulting code. It is not the purpose of the present research to be an exercise in computer programming, but rather an investigation of signal (and particularly image) processing procedures that appear to lend themselves to efficient implementations. It is for this reason that details of the computer programming aspects of this work are left to the appendices which contain both program documentation and complete program listings. In the present chapter the computer programs are described strictly from an image processing point of view.

The program that implements the experimental system

(system a of figure 3.1) is entitled PROC12. The one that implements the conventional system, used as a control in these experiments, is PROC13. Both of these programs perform each of the various processing tasks over entire pictures before proceeding on to the next function. That is, if a picture is to be compressed as part of the processing, the entire picture is compressed before the next required processing function is performed. It could well be that an efficient hardware implementation of these algorithms would be based not on a picture (or frame) sequential architecture but rather on a line sequential one. In this case pictures would be processed one line at a time. In fact such an architecture lends itself nicely to a "pipelined" system where several operations are performed simultaneously. There is however no fundamental algorithmic difference between this and the present implementation; only one of programming detail. The processing modules pass data from one to the other through a series of files on magnetic disk drives called picture files. Within each processing module the image is operated on one picture row at a time.

The sources of the picture data were various positive transparencies and photographic prints. These were scanned into the image processing system with an Image Digitizer manufactured by the EIKONIX Corporation. This device makes use of an illumination system, an objective lens, and a self-scanned photodiode array mounted on a precision stepping stage to convert optical transmittance (or reflectance in the

case of the reflection samples) into a computer-compatible PCM digital image. These initial PCM images will be referred to as source images. The images were chosen for their different subject types, wide tonal distributions, and varied spatial frequency content.

Before being processed by the two experimental systems, the source pictures were digitally blurred. The degrading consisted of filtering the source pictures with a 15-element blurring kernel. The actual filtering operation was affected by the same subroutine that performed the restoration filtering within both of the image processing systems. The blurring filter had a Gaussian impulse response with a standard deviation of 2 pixels. The Gaussian form was chosen to be, in some sense, typical of many image degradations; particularly of the impulse response of systems in which degradations from several sources occur. The width of the blur was chosen somewhat arbitrarily so as to produce a clearly noticeable degradation of the image without over-taxing the design effort of the restoring filter.

The blurring filter was constructed by program GAUS1D, which also computed the restoring filter used by both the experimental and the control image processing systems. The restoring filter was computed in the spatial frequency domain according to equation 4.4.

$$H(f) = B^*(f)/\{|B(f)|^2 + k\} \qquad 4.4$$

In equation 4.4, H is the restoring filter, a function of spatial frequency, f, B is the blurring filter computed as the Fourier transform (actually DFT) of the Gausian impulse response, and the (*) indicates complex conjugation. The constant, k, was used to approximate the inverse signal to noise term in the Wiener filter of equation 2.15. The program GAUSlD writes both the blurring and the restoring filters to data files that are read, as required, by the blurring and processing programs. The blurred versions of the source pictures are referred to as input pictures, since these are the input data to both the experimental and the control image processing programs.

Both of the programs begin by compressing the input picture using a one-dimensional DPCM algorithm. A flow chart of the compressor is shown in figure 4.1.

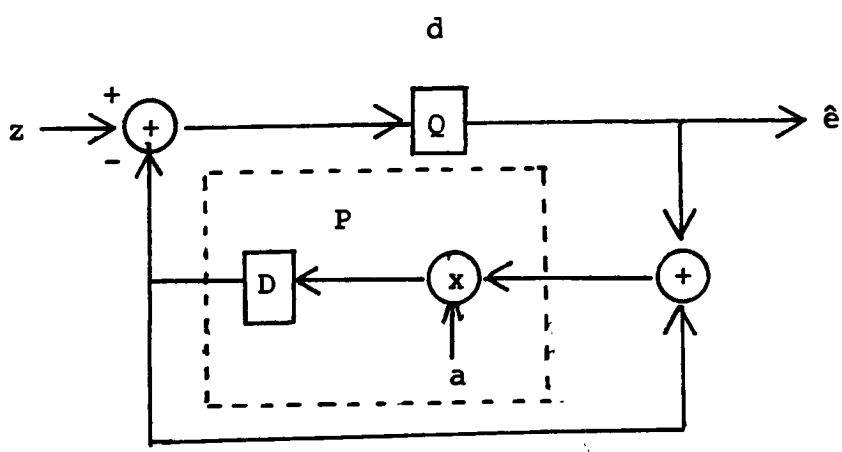

Figure 4.1. Block Diagram of DPCM Compressor showing z, input, ê, output, Q, uniform step size quantizer with step size d, P, predictor consisting of prediction multiplier, a, and single sample delay, D.

The input to the compressor is the sampled image data (PCM image) z, the output is the quantized error signal $\hat{e}$. The prediction coefficient, a, and quantizer step size, d, are parameters that are held constant for a particular experimental run of the program, but that are varied in a controlled way as part of the experimental design. The predictor, P, consists of a multiplication by the factor a, and a single sample delay implemented by D.

As described in chapter 3, the error signal is first formed by subtracting from the input sample the prediction for that sample. The error signal is calculated as a floating point number and then quantized to an integer containing only a small number of bits. This is transmitted as the output, $\hat{e}$. The quantized error signal is then summed with the predicted value to form the reconstructed pixel value for the current sample. This in turn is scaled by the prediction coefficient, a, and delayed one sample period to form the prediction for the next sample in the row. To start the process going the predicted value for the first point in each row is taken as zero.

After the compression stage the two systems differ somewhat. In the case of the experimental system, a column of correction terms is computed, the compressed image is filtered, and then the compressed and filtered picture is corrected and reconstructed (decompressed) in one step. In the case of the conventional system however the image is immediately decompressed (no correction is necessary) and

then filtered.

In the case of the experimental system the correction terms are computed according to equation 3.27. These values (one for each row of the picture) are stored so that they can be summed with the first reconstructed pixel in each row upon decompression of the image.

After computation of the correction terms the DPCM compressor output is filtered with a 15-element non-recursive filter. The particular kernel used in the filter is computed prior to picture processing and is an estimate of a Wiener filter with the inverse signal to noise ratio term replaced with a single parametric value for all frequencies. This parameter is used to alter the "aggressiveness" of the filter. The filter is normalized to unit gain for D.C. inputs and is applied in the spatial domain with code equivalent to equation 3.28 with M, the size of the filter kernel equal to 15. That is, when the input to the filter is the quantized DPCM error signal at sample position j across the row, and the filter kernel values are h_i (i=1,2,...,15), the output values are computed according to equation 4.1.

$$\hat{\hat{e}}_j = \sum_{i=1}^{15} h_i \, \hat{e}_{j+8-i} \, . \qquad 4.1$$

Input samples outside the domain of the picture are assumed

to have values of zero. The additional term of 8 in the index of the input samples is included to maintain the zero phase relationship between filter input and output.

Finally, in the experimental system the filtered DPCM error signal ($\hat{\hat{e}}$ of equations 3.32 through 3.35) is decompressed. The block diagram of the decompressor is shown in figure 4.2.

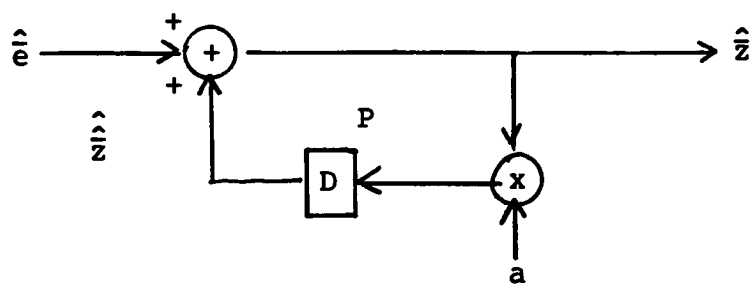

Figure 4.2 Block Diagram of DPCM Decompressor
$\hat{\hat{e}}$, compressed signal input, a_k prediction coefficient
D, single sample delay, and $\bar{\bar{z}}$, decompressed output.
P is the predictor

Algebraically the output of the decompressor at sample position j is simply

$$D_j = C_j + aD_{j-1} \quad . \qquad 4.2$$

In the case of the experimental system the predicted value for the first pixel in the line is set equal to the correction term computed previously, thus affecting the correction and decompression simultaneously.

$$D_j = \begin{cases} C_j + aD_{j-1} & j>1 \\ C_j + \text{Correction} & j=1 \end{cases} . \qquad 4.3$$

For the conventional system the processing is only slightly different. After the compression, the image is decompressed using the same method as the experimental system, but with the correction terms of equation 4.3 all having values of zero. Next, the decompressed picture is filtered using equation 4.1. All experimental parameters including a in the predictor, d in the quantizer, and the 15 filter coefficients are identical for the experimental and the control systems.

4.2 Objective Comparisons

In order to analyze the differences between the pictures produced by the experimental system and the control system, a program entitled DIF has been written. This program creates a third picture consisting of the algebraic difference between corresponding pixels of the outputs from the two image processing systems and performs various analyses on the resulting error picture. After the creation of the error picture its probability density function (pdf) is constructed by counting the number of times each of the 256 possible data values occurs and dividing by the total number of pixels in the picture. Next, the pdf is used to compute various error statistics including the mean, variance, and zero memory entropy. These are computed according to equations 4.5 through 4.7.

$$\mu_E = \sum_E E\ p(E) \ . \qquad 4.5$$

$$\sigma_E^2 = \sum_E E^2\ p(E) \ - \ \mu_E^2 \ . \qquad 4.6$$

$$H_E = \sum_E p(E)\ \log_2[p(E)] \ . \qquad 4.7$$

In equations 4.5 through 4.7 the E are the various possible values in the range of the processing error introduced by commuting the decompression and filtering operations. μ, σ^2, and H represent the mean, variance, and zero memory entropy respectively. The sums are taken over all possible values of E.

Following the computation of these summary statistics, the error picture is analyzed to find its mean and mean squared values as functions of the position across the line. For the error picture E(k,j) with column index j and row index k, these values are computed according to equations 4.8 and 4.9.

$$\mu_{E(j)} = (1/N) \sum_k E(k,j) \ . \qquad 4.8$$

$$\sigma^2_{E(j)} = (1/N) \sum_k E^2(k,j) - \mu^2_{E(j)} . \quad 4.9$$

These values are useful for testing the validity of equations 3.51 through 3.54.

4.3 Experimental Plan

To summarize, table 4.1 gives the parameters that were varied as part of the experimental plan.

Table 4.1
Definitions of Controlled Parameters

Symbol	Definition
a	DPCM Prediction Coefficient
d	DPCM Quantizer Step Size
k	Inverse Signal to Noise Ratio Approximation

Values of 1.0, 0.9, and 0.8 were chosen for a. These give enough range to test equations 3.51 through 3.54 and to measure subjectively any degradration of the image quality due to the use of the experimental algorithm. A quantizer step size of 2.0 was chosen so as to yield reasonable data compression ratios without introducing an unmanageable amount of noise. Although the value of k was held constant between the experimental and the control systems, it was varied from picture to picture to account for the differences in the

power specta of the source images.

Two pictures were chosen for processing; a low frequency head and shoulders portrait and an aerial image containing large amounts of power at high spatial frequencies. Since each picture was processed with three different values of a for the experimental system, and again with the same three values of a in the control system, twelve experimental runs were performed.

To compute k for each picture, it was first assumed that the total noise power for each picture was only that produced by the quantizer. For d=2, the mean squared noise so introduced is simply $2^2/12$, or about 0.333. Since the quantizing noise samples are independent, it can be assumed that the noise is spectrally flat, and for purposes of computing the filter the noise power at each spectral sample, Φ_n, is given by equation 4.10.

$$\Phi_n = 0.333/N \ . \qquad 4.10$$

In equation 4.10, N is the number of columns in the picture.

The spectrum of the signal input to the blurring filter, B, was measured directly for each of the source pictures. This was done simply by computing the square of the modulus of the Fourier transform for each row of the picture and then taking the average over all the rows.

The value of k was then computed according to equation 4.11 at each spatial frequency.

$$k = \Phi_n/\Phi_z = 1/(3N\Phi_z) \quad . \qquad 4.11$$

Since the value of k so computed was very nearly constant at high frequencies, a single value was chosen to compute the filter for each of the two pictures. In the case of the low frequency content portrait a value of k of 0.035 was chosen, in the case of the aerial scene a value of 0.004 was used. The spatial and the spatial frequency domain representations of the filters used are shown in figures 4.3 and 4.4.

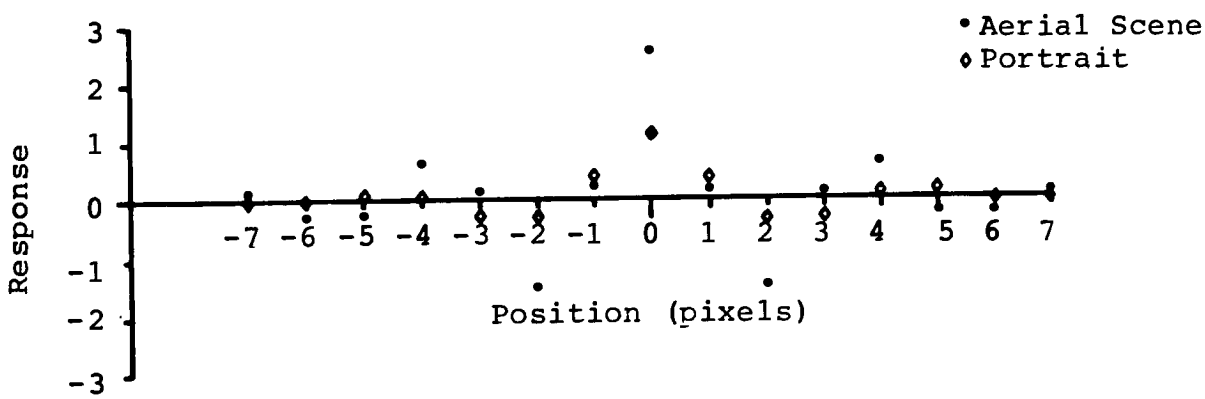

Figure 4.3
Spatial Domain Representation of Deblurring Filters

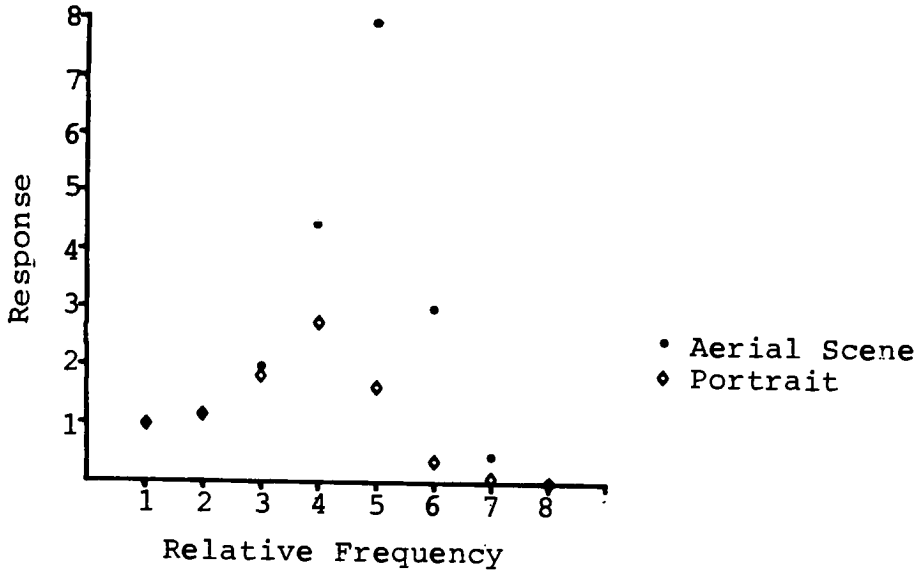

Figure 4.4
Spatial Frequency Domain Representation
of Deblurring Filters

After digitizing, the source pictures were characterized by finding their pdf's and various summary image statistics including their means, rms values, and zero memory entropy. The source pictures are shown in figures 4.5 and 4.6, the pdf's are shown in figure 4.7, and the summary image statistics are given in table 4.2. These data are presented for reference only. The objective and subjective evaluations were performed comparing the images output from the two image processing systems to each other. There is no doubt that with more attention paid to the filter design the output images could be made to match the sources more closely. This is not the purpose of the present work; rather, it is to match the outputs of the two image processing systems to each other.

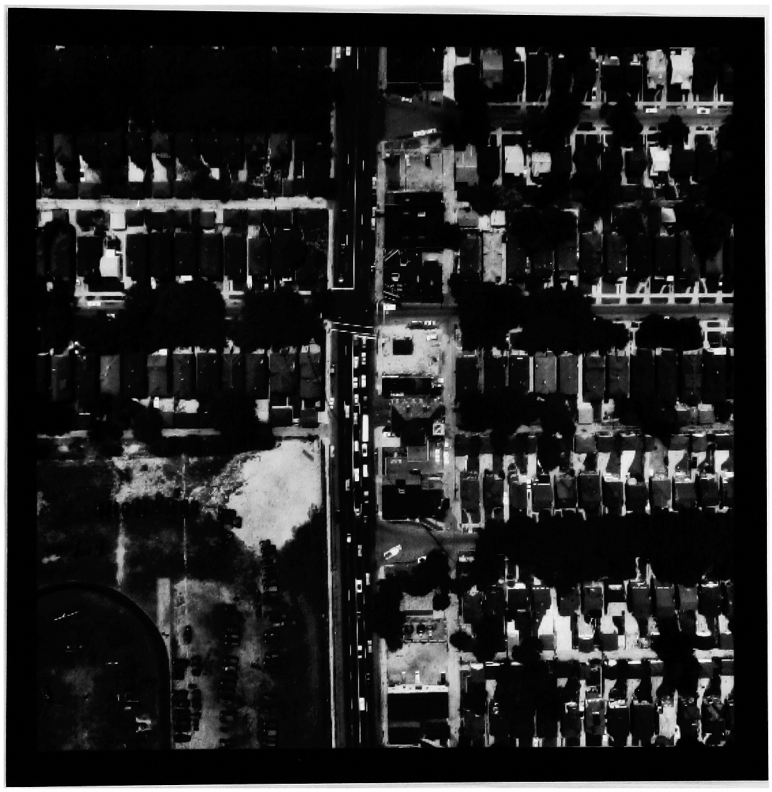

Figure 4.5
Source Picture - Aerial Scene

Figure 4.6
Source Picture - Portrait

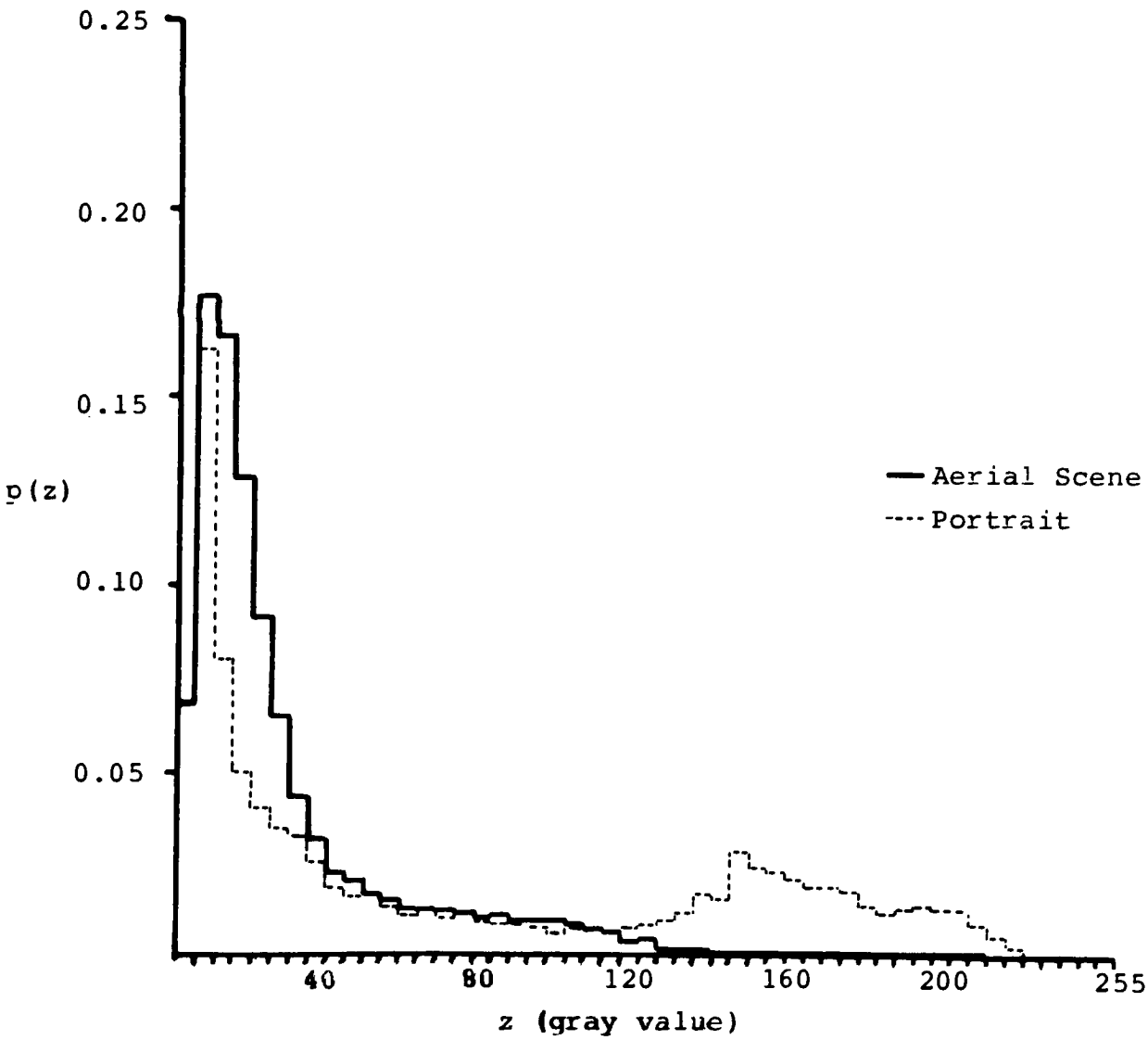

Figure 4.7
Probability Density Functions of Source Pictures

Table 4.2
Summary Image Statistics for Source Pictures

Picture	Mean	Standard Deviation	Zero Memory Entropy
Aerial Scene	31.4	33.0	6.26
Portrait	76.0	71.7	7.18

4.4 Subjective Evaluations

Because the image processing phase of this work was divided so clearly by the image processing algorithm employed, the natural choice of an evaluation method was that of a paired comparison. Each of the six images processed through the control system (two pictures processed with three different values of the parameter a) was compared to the corresponding pictures processed with the experimental system. The pairs of images were presented for inspection simultaneously and in a random order. The judges were asked which of the two pictures was preferred and the results tabulated. The parameter being tested is the proportion, p, of judges preferring the image produced by the control system for each treatment of each picture. The hypothesis can be simply stated as follows:

$$H_0: \quad p = p_0 = 0.5$$

$$H_1: \quad p \neq p_0$$

Since the experiment consists of a repeated set of independent Bernoulli trials, the statistic, p, has a

binomial probability density. The critical region (that which requires the null hypothesis, H_0, to be rejected) is simply that section of the domain of opportunity below (in the case of the lower tail) and above (in the case of the upper tail) in which a fraction $\theta/2$ of the population falls. θ is the risk of a type I error so that the statement is made with a degree of confidence $1-\theta$. The binomial probability is well tabulated in a variety of sources.[30] For the sample size of 12 observers employed in these experiments, and for $\theta = 0.05$, the null hypothesis must be rejected whenever the number of judges preferring the control system is 0, 1, 2, 10, 11, or 12. These values will be employed in the results section of the following chapter.

4.5 Image Reproduction

The images processed, evaluated, and presented in this work are all 512-by-512 pixels in size. The original for the aerial scene was a portion of a transparency duplicate, that for the portrait was a reflection photographic print. Both of the images were scanned using image digitizers manufactured by the EIKONIX Corporation. The sampling apertures were about one-quarter the size of the sample spacing in both dimensions. The image digitizers produce 8-bit PCM images that are linear in transmittance (in the case of the transmitting samples) and reflectance (in the case of the reflecting samples).

After processing, the images were written with a drum type laser beam recorder (LBR) also manufactured by the

EIKONIX Corporation. This instrument writes, onto a piece of photographic film, a negative using a Helium Cadmium laser and an acousto-optic modulator. The resulting negatives were contact printed onto a standard photographic paper to produce positive reflection prints. The LBR is equipped with an 8-bit-in 12-bit-out look-up table, with which a linearization function may be applied to the data being written. This table was loaded with the inverse of the combined LBR, photographic negative film, and photographic paper response characteristic so as to produce a nearly linear system tone reproduction. The output tone reproduction is shown in figure 4.8. Gray scales were added to all of the pictures in order to check the reproduction characteristics of the system. The raster frequency of the LBR was set to approximately 5 lines per millimeter when writing the pictures used in these experiments.

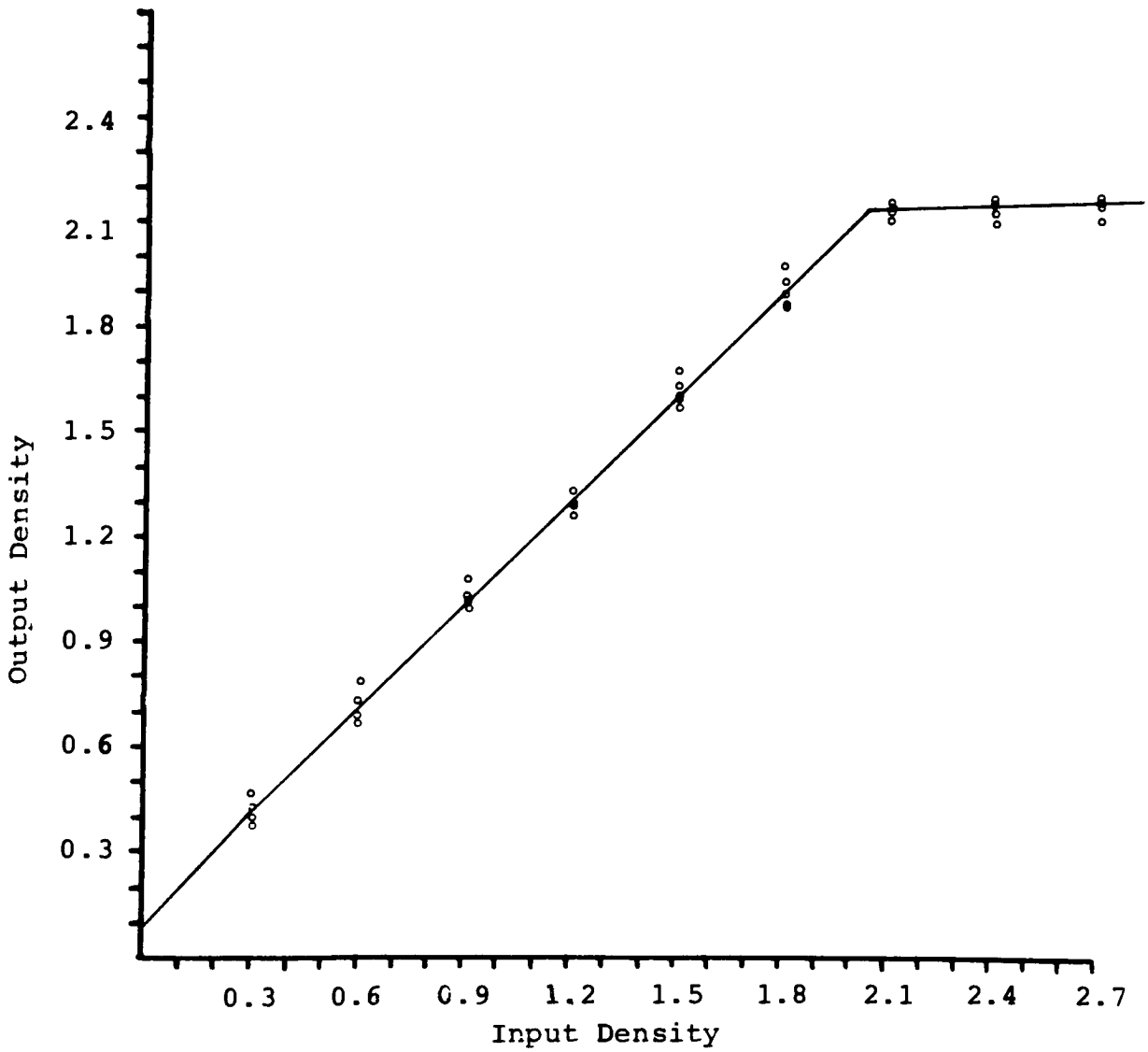

Figure 4.8
Output System Tone Reproduction

5. RESULTS AND DISCUSSION

5.1 Effects of Noise

The images processed with the experimental and the control systems are shown in figures 5.1 through 5.6. Each figure contains the resulting pictures from both systems for a single value of the prediction coefficient, a. The product of the experimental system is shown in each case on the top. That of the control system is on the bottom.

It is immediately apparent by examining figures 5.1 and 5.4 that the horizontal streaks are highly objectionable in the experimentally processed pictures. Note that these lines get more distinct across the width of the images. As a matter of fact, very careful examination of the extreme left-hand sides shows regions in each picture where the effect is almost nonexistent. It was shown in chapter 3 that the mean squared error between images produced by the two systems increases linearly across the width of the picture when a, the prediction coefficient, has a value of unity. This is because the noise samples produced by the filter operation are accumulated in the integrator that constitutes the decompressor. Pixels on the right side of the pictures contain the sums of many noise samples, those on the left side only a few. Thus the horizontal streaks get progressively worse toward the right side of the pictures.

Figure 5.7 is a set of plots of the mean squared error between the experimentally processed images and those processed with the control system as functions of the picture column index, j. According to equation 3.54, this plot should be linear with an intercept of zero and a slope of $d^2/12$. In this case d had a value of 2 so the slope should be 1/3. The line representing that model is plotted on the same set of axes as the data in the figure. Note that the data fit the model quite well for both pictures, though some departure is apparent for large values of j. Several causes for the limited departure are worth discussing. First, the mean squared error itself is a noisy quantity that is subject to sampling error. The greater the mean squared error, the greater its own standard error (the variance of the variance if you will) thus the greater the departure for larger values of j. Second, and perhaps more important, is the fact that the lines in each picture are not independent ensembles. If instead of using different rows of the same picture as experimental samples, random rows of many different pictures were chosen, then one would expect a closer fit to the model.

When a was reduced to a value of 0.9 (see figures 5.2 and 5.3) the horizontal lines became far less objectionable. In fact, in the case of the aerial scene they are apparent only on very close inspection. That these artifacts are less objectionable in the aerial scene than in the portrait is probably due only to the fact that the picture is busier.

Figure 5.1
Portrait Processed Conventionally (top) and Experimentally (bottom) for a = 1.0.

Figure 5.2
Portrait Processed Conventionally (top) and
Experimentally (bottom) for a = 0.9.

Figure 5.3
Portrait Processed Conventionally (top) and
Experimentally (bottom) for a = 0.8.

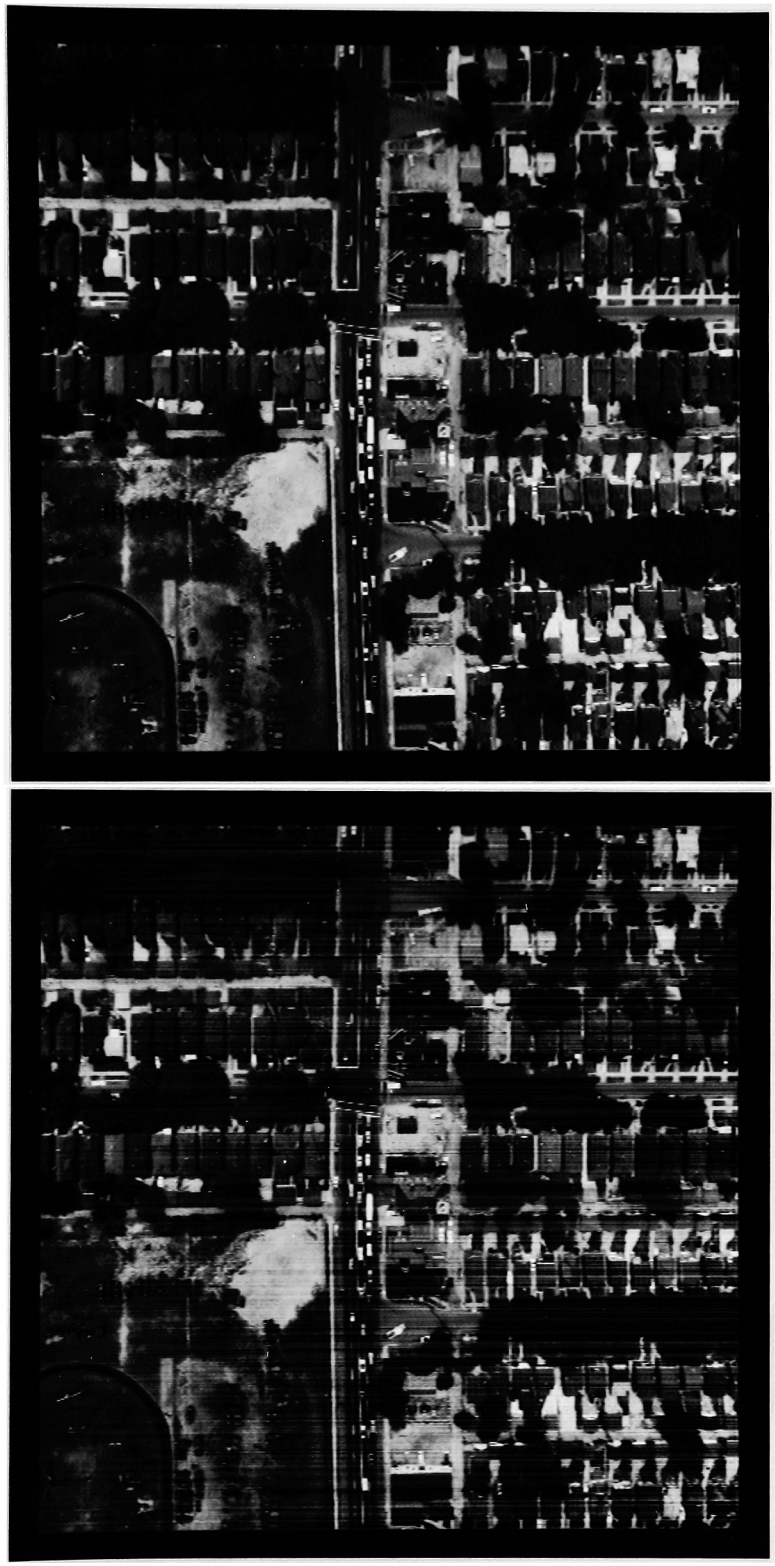

Figure 5.4
Aerial Image Processed Conventionally (top) and Experimentally (bottom) for a = 1.0.

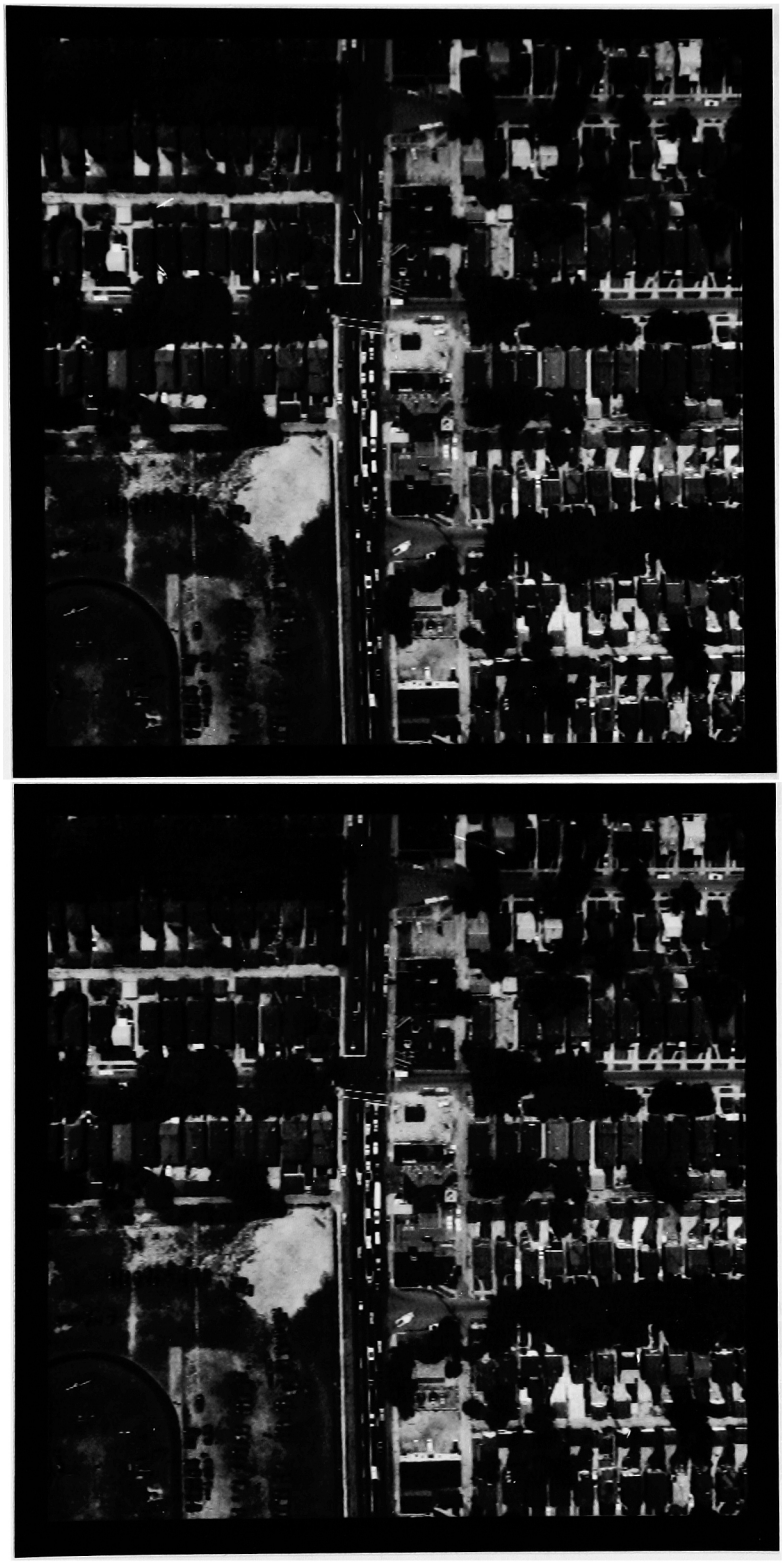

Figure 5.5
Aerial Image Processed Conventionally (top) and Experimentally (bottom) for a = 0.9.

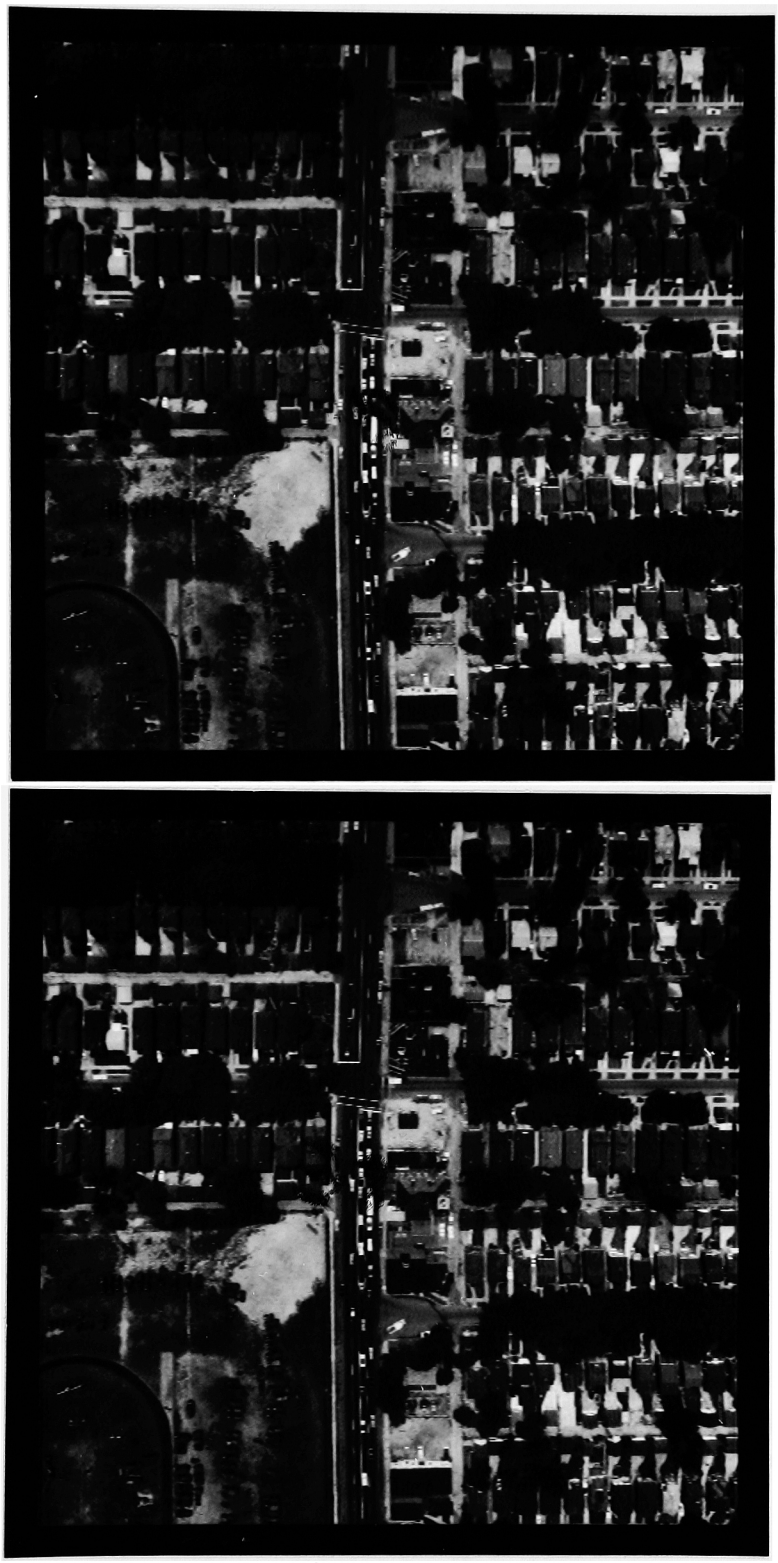

Figure 5.6
Aerial Image Processed Conventionally (top) and Experimentally (bottom) for a = 0.8.

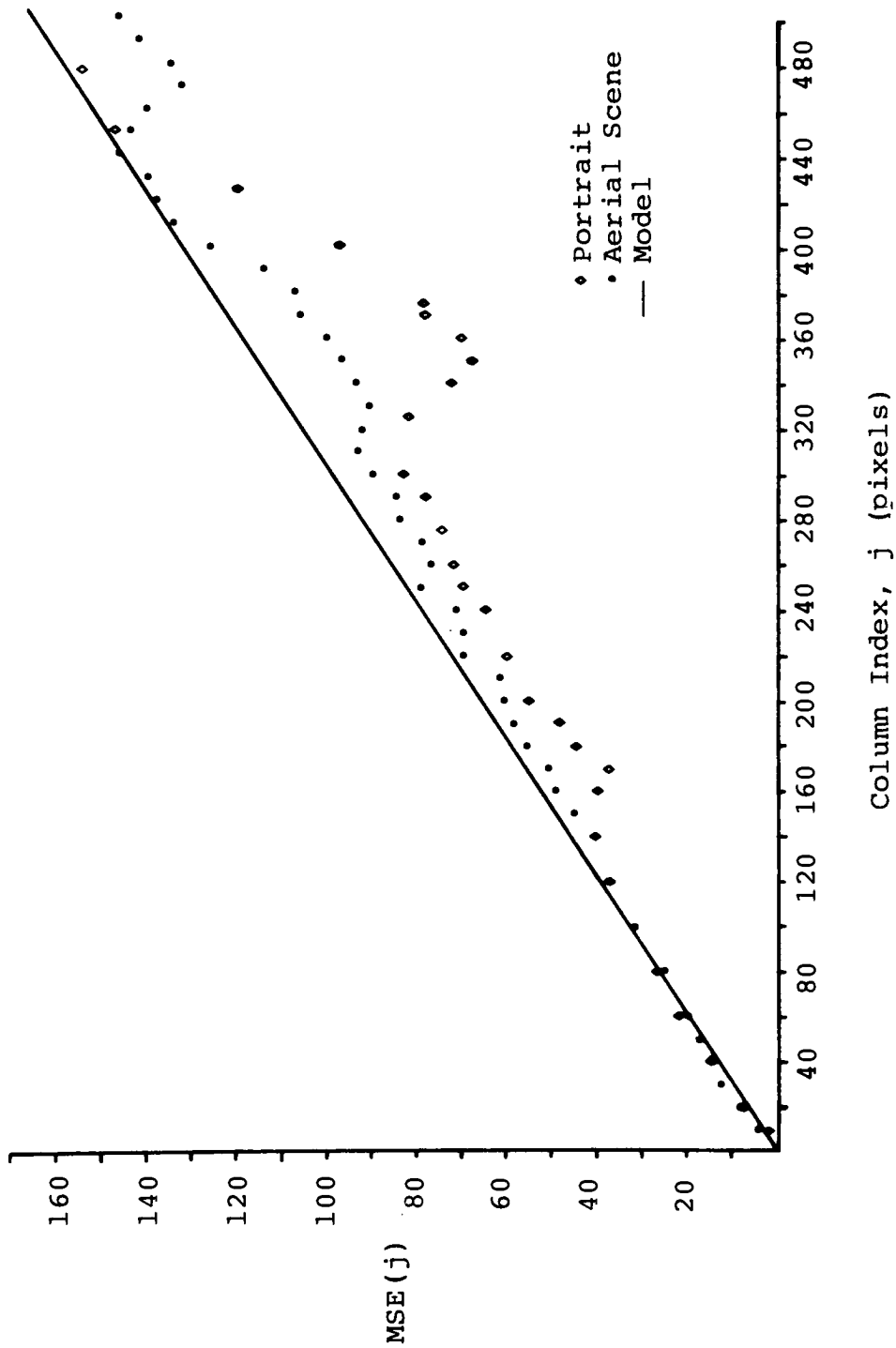

Figure 5.7
Mean Squared Error (MSE) Between Experimentally and Conventionally Processed Images As Functions of Column Index for a = 1.0

The signal to noise ratio is much higher at high spatial frequencies in the aerial scene than in the portrait. Further reduction of a from 0.9 to 0.8 reduces these artifacts still more. In the case of the aerial scene of figure 5.3, the horizontal lines are almost undetectable under normal viewing conditions. Strongly correlated horizontal noise components are still visible in the portrait at this reduced value of a, but this result could be acceptable in some applications.

The observed reduction in horizontally correlated noise is predicted by equation 3.61 where the error between the conventionally processed and the experimentally processed pictures was found to depend on the quantity $1/(1-a^2)$. In figure 3.3 it was also shown that the limiting value of the MSE should be approached very quickly. Table 5.1 gives average values of the mean squared error between the outputs of the two systems for both test pictures and both values of a. Also tabulated are the theoretical values predicted by equation 3.58.

Table 5.1
Averaged Mean Squared Errors

Picture	a=0.9	a=0.8
Aerial Scene	2.95	2.13
Portrait	2.05	1.14
Theoretical Value	2.09	1.26

Notice that the error terms for the aerial scene are greater than those predicted by the model. However, the error for a equal to 0.9 is less than that for a equal to 0.8 as would be expected in all cases. It is interesting to note that the differences between the mse terms for each value of a are 0.82, 0.91 and 0.83 for the aerial scene, the portrait, and the theoretical values respectively. This would indicate that the additional noise components introduced by increasing a from 0.8 to 0.9 are nearly constant and equal to that predicted by equation 3.58, even though there may be some noise components that are not accounted for in the present model.

5.2 Results of the Subjective Evaluations

A summary of the results of the subjective evaluations is presented in table 5.2.

Table 5.2
Results of the Subjective Evaluations
For 12 Judges

Picture	Prediction Coefficient a	Number of Judges Preferring Control k	Significant at 95% Confidence
Aerial Scene			
	1.0	12	Yes
	0.9	10	Yes
	0.8	7	No
Portrait			
	1.0	12	Yes
	0.9	11	Yes
	0.8	8	No

Notice that for both scenes the judges were unanimous in their preference of the conventional image processing system for a=1.0 and nearly so for a=0.9. The results were mixed with a=0.8. The last column in table 5.2 indicates whether or not the observed number of judges preferring the control system, k, is significant at a 95 percent level of confidence. An entry of "yes" indicates that the null hypothesis, as stated in section 4.4, should be rejected and that the observed differences in the images can be assigned to differences between the image processing systems. An entry of "no" indicates a failure to reject the null hypothesis; thus assigning any observed preference of one image over the other to chance alone. Thus with a=0.8 the systems have been judged to be equivalent.

5.3 Effects on the Compressor Performance

It is clear from the results presented in the last section that reduction of the value of the prediction coefficient reduces the noise and the objectionable artifacts by substantial amounts. It is well worth considering the cost that might be paid in reduced performance of the compressor.

Since a signal with zero memory entropy H_0 can be encoded using approximately H_0 bits per sample with straightforward coding techniques, this is a useful measure for estimating the number of bits required to represent the output of the DPCM compressor. Table 5.2 shows the values of H_0 for the input pictures and the DPCM error signals, the $\hat{e}$

of chapters 2, 3, and 4 for both pictures for the various values of a.

Table 5.3 Zero Memory Entropy for Input and Compressed Pictures In Bits per Pixel (BPP)

Picture	Input	Compressed Images		
		a=0.8	a=0.9	a=1.0
Aerial Scene	6.24	3.59	3.37	3.34
Portrait	7.17	4.16	3.62	2.70

In the case of the aerial scene there was a negligible loss in compressor efficiency when the prediction coefficient was reduced from 1.0 to 0.9, and only a slight loss when it was further reduced to 0.8. In the case of the portrait however, the compressor performance is a strong function of a. Note that the initial compression of the portrait (a=1.0) was considerably more significant than that of the aerial scene. In the case of the portrait the entropy was reduced by 4.47, 3.55, and 3.01 BPP for a equal to 1.0, 0.9 and 0.8 respectively. In the case of the high frequency aerial scene the savings were only 2.90, 2.87, and 2.65 BPP for the same three values of a. In short, the performance of the compressor for the portrait decreased with decreasing a because the performance was so much better in the first place. The high frequency content of the aerial scene

corresponds to a relatively narrow autocorrelation function so that many pixels are poorly predicted; even with small values of a. Because the gray levels of the portrait are slowly varying they are predicted very well with large values of a, and poorly with small.

6. CONCLUSIONS

A digital image processing system has been simulated that contains a one-dimensional differential pulse code modulation (DPCM) compressor, a one-dimensional linear filter, and a DPCM decompressor. An experimental system has been simulated that reverses the order of the filtering and decompression functions.

It has been found that the filter and decompression functions can be commuted with no degradation of the image quality if the prediction coefficient, a, in the compressor is set equal to 0.8 and if a simple correction term is computed and applied. For a=0.9, there is a noticeable degradation in the quality of the pictures, but the experimental system may produce results that are acceptable for some applications. With a=1.0, the commutation of the decompression and filtering operations causes severe streaks in the output, which get progressively worse across the width of the pictures. These artifacts cause the pictures produced by the experimental system to be unacceptable.

The most objectionable errors between the two systems are caused by a final rounding-to-integer operation at the end of the filter. In the case of the conventional system this results in a single sample of noise at each pixel position. This sample is not correlated with any other noise

samples. In the case of the experimental system, however, since the samples of quantizing noise introduced by the filter are added before the decompressor, they accumulate there thus increasing the errors on final output. The result is similar to that produced by a DPCM communication system with a noisy channel.

It might be feasible, in some applications, to combine the filtering and decompressing functions into a single process. If neither storage nor transmission of the compressed and filtered data is required, there is no need to integerize the output of the filter; rather, it could be passed directly on to the decompressor with full precision. This would eliminate the objectionable artifacts regardless of the value chosen for the prediction coefficient, a.

The objective measures of error (noise) between the two systems have been modeled and the data shown to fit the model quite well. The model predicts noise increasing with the value of a. Also, for a=1.0, the noise increases across the width of the picture. These results predict the observed degradation in image quality that was measured with a paired comparison subjective evaluation by twelve independent observers.

The performance of the compressor was somewhat reduced with the lower values of a, but useful compression rates were achieved for all three values tested. The loss of compression efficiency was less severe for a busier scene than for one with less high frequency content.

The experimental system shows promise for being a more efficient implementation of the basic compression, filtering, and decompression functions with little, if any, loss in image quality and only moderate reduction of compression rates.

7. RECOMMENDATIONS FOR FUTURE WORK

7.1 Extensions to Two-Dimensional Processing

The major drawback of the current work is its limitation to one-dimensional processing of pictures, both for the compression operations and for the filtering. In the case of the compression, the one-dimensional processing affects the amount of data reduction achievable. In the case of the filtering it affects the utility of the operation. Other authors have shown[31] that two-dimensional predictors can yield increased compression rates of 20 to 30 percent over their one-dimensional counterparts. Since nearly all real world image degradations are two-dimensional in nature, a two-dimensional enhancement operator is necessary.

In order to extend this work to two-dimensional processing it will first be necessary to build models analogous to those developed in chapter three of the present work. The predominant source of objectionable noise has been found to be that added by the filter. This is outside of the feedback loop of the compressor and on decompression the error terms build up. In the case of the one-dimensional compressor, it was a fairly easy task to model these effects upon decompression for values of the prediction coefficient between zero and one and to reduce the model algebraically to a simple expression. In the case of the two-dimensional

compressor, the model becomes somewhat more complicated. A flow chart of the two-dimensional decompressor is shown in figure 7.1.

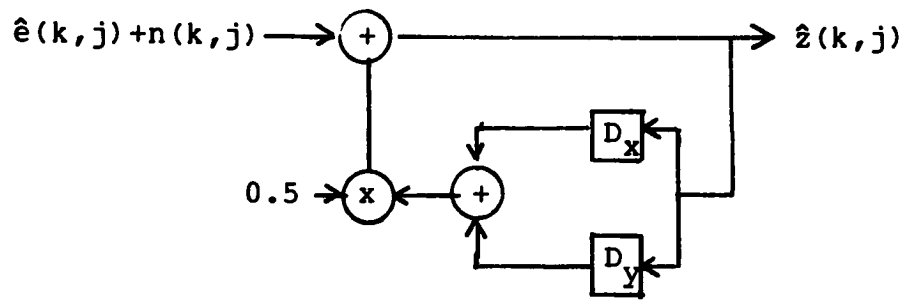

Figure 7.1 Two-Dimensional DPCM Decompressor
D_x and D_y single sample delays in the x and y dimensions
Prediction coeffieicients are 0.5

In figure 7.1 the input is the sum of the quantized DPCM error signal, ê, and a noise sample, n, at each pixel location. The output is $\hat{z}(k,j)$ which is given by equation 7.1.

$$\hat{z}(k,j) = \sum_{m=0}^{k-1} \sum_{i=0}^{j-1} 0.5^{m+i} \binom{m+i}{i} [\hat{e}(k-m,j-i)+n(k-m,j-i)] \qquad 7.1$$

In equation 7.1, $\binom{m+i}{i}$ indicates a binomial coefficient representing the number of combinations of i elements that can be taken from a set of size m+i. The resulting noise component of the output image is simply the noise component of equation 7.1. Let this be N(k,j), which is given by equation 7.2.

$$N(k,j) = \sum_{m=0}^{k-1} \sum_{i=0}^{j-1} 0.5^{m+i} \binom{m+i}{i} [n(k-m,j-i)] \qquad 7.2$$

Equation 7.2 can be rewritten in several more compact forms, none of which lends itself to algebraic evaluations, only numerical ones. It is important to note that because of the fraction 0.5 raised to an increasing power, the noise contributions from preceding samples tend to decay with distance from the current sample. It has not been proved that the decay is more rapid than the increase of the binomial coefficient term, but there is some promise that useful results could be obtained with such an algorithm. The decay and thus the magnitude of the noise term, N(k,j), would be even more rapid if the prediction coefficients were reduced to values below 0.5. Any further development of this model should allow for the change of these parameters as did the model developed in section 3.3.

It may be that no changes to the current models are required to accommodate the change to a two-dimensional spatial filter. The most predominant source of error in the experimental system is not the filter itself but rather the accumulation of many noise samples in the decompressor. In the case of a two-dimensional filter only one noise sample is added to each compressed pixel value as is the case with the one-dimensional filter. The appendix contains a subroutine to apply the two-dimensional filter (FILT2D) to an image that

can easily be replaced for the one-dimensional filter currently applied. It is suggested that the required two-dimensional filters be constructed and the simulations run with the double-dimensional filter.

7.2 Calculations with Coded Data

In the current work no attempt was made to code the compressed images. Representing the quantized DPCM error signal with fixed 8-bit-long words simplified the storage operations and allowed straightforward calculation of the filter equations using general purpose computers. For the compression to be effective however the DPCM compressor output must be encoded with a variable word length code. To avoid the requirement that the image be decoded, filtered, and then recoded, it is necessary to develop methods of performing calculations (particularly multiplications and additions) on the variable word length numbers. This is an algebra problem of potentially general application, which may already have a solution from another field of research.

7.3 Other Applications

Only one application of a linear filter has been investigated here--that of a sharpening, or enhancement filter. Other similar calculations might lend themselves to application on compressed image data. The sharpening kernel, for example, might be replaced with a matched filter (such as a character or object shape) for applications involving pattern recognition or object location. Also, various sizing and scaling algorithms might be more efficiently implemented

in the compressed rather than the decompressed domain.

8. LIST OF REFERENCES

[1]A. Habibi, "Introduction--Special Issue on Image Bandwidth Compression," IEEE Transactions on Comunications COM-25 (November 1977):1249-1251.

[2]R. Gonsalez and P. Wintz, Digital Image Processing (Reading: Addeson-Wesley, 1977), 1-5.

[3]N. S. Jyant, "Digital Coding of Speech Waveforms:PCM, DPCM, and DM Quantizers, "Proceedings of the IEEE," 62 (May 1974):611-632.

[4]A. Habibi, "Comparison of n'th Order DPCM Encoder with Linear Transformations and Block Quantization Techniques," IEEE Transactions on Comunications COM-19 (December 1971):948-956.

[5]C. E. Shannon, "A Mathematical Theory of Communication," Bell System Technical Journal 27 (July and October 1948): 379-423 and 623-656.

[6]Ibid.

[7]Ibid.

[8]D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes" Proceedings of the IRE 40 (September 1952):1098-1101.

[9]R. Fano, "Technical Report No. 65--The Research Laboratory of Electronics, M.I.T."

[10]A. Habibi, "Comparison of n'th Order DPCM Encoder with Linear Transformations and Block Quantization Techniques."

[11]A. Habibi, "Hybrid Coding of Pictorial Data," IEEE Transactions on Comunications COM-22(May 1974):614-624.

[12]A. Habibi and P. Wintz, "Image Coding by Linear Transformation and Block Quantization," IEEE Transactions on Comunications COM-19 (February 1971):50-62.

[13]L. E. Franks, "A Model for the Random Video Process," Bell System Technical Journal 45 (April 1966):609-630.

[14]J. B. O'Neal Jr., "Predictive Quantizing Systems (Differential Pulse Code Modulation) for the Transmission of Television Signals," Bell System Technical Journal 45 (May 1966):689-719.

[15]N. Ahmed, T. Natarjan, and K. Rao, "Discrete Cosine Transform," IEEE Transactions on Computers C-23 (January 1974):90-93.

[16]O'Neal, "Predictive Quantizing Systems (Differential Pulse Code Modulation) for the Transmission of Television Signals."

[17]Jyant, "Digital Coding of Speech Waveforms:PCM, DPCM, and DM Quantizers."

[18]Ibid.

[19]O'Neal, "Predictive Quantizing Systems (Differential Pulse Code Modulation) for the Transmission of Television Signals."

[20]J. C. Dainty and R. Shaw, Image Science (New York: Academic Press, 1974), 204-215.

[21]Gonzalez and Wintz, Digital Image Processing, Section 5.1.

[22]M. A. Kriss, "Chapter 21," in The Theory of the Photographic Process, ed. T. H. James (New York: Macmillan, 1977).

[23]J. W. Goodman, Introduction to Fourier Optics (New York: McGraw Hill, 1968), Section 2.7.

[24]Ibid.

[25]R. Bracewell, The Fourier Transform and Its Applications (New York: McGraw Hill, 1965).

[26]Gonzalez and Wintz, Digital Image Processing, Section 5.5.

[27]J. Freund, Mathematical Statistics, (Englewood Cliffs: Prentice-Hall, 1971), 95.

[28]Ibid.

[29]L. B. W. Jolley, Summation of Series, (New York: Dover, 1961), 2-3.

[30]Freund, Mathematical Statistics, 425-429.

[31]J. O. Limb, "Entropy of Quantised Television Signals," Proceedings of the IEE 115(January 1968):16-20.

9. APPENDIX 1

SOURCE CODE LISTINGS

This appendix contains, with only a few exceptions, the FORTRAN source code listings for all of the programs and subroutines used by the experimental and control image processing systems. The programs are generously sowed with comment statements so that little effort is made here to supply additional documentation. The source code should be readily understandable by any reader with a moderate undestanding of the FORTRAN programming language.

Four particular subroutines are not included in this appendix. They are system dependent input/output (I/O) routines that are used to open, read, write, and close the picture files. The names and argument lists are contined in tabel A1.1.

In the table the variable LUN refers to a logical unit number on which the operation is to occur, FILE is a valid file name, NCOLS and NROWS are the number of columns and rows, respectively, in the picture, and AGE has either the value 'NEW' or 'OLD' depending on whether the picture already exists or whether it is being created. IROW1 And IROW2 indicate the first (IROW1) and last (IROW2) rows of the picture to be read or written, and BUFFER is the address to

which the data is read or written.

TABLE A1.1

PICTURE I/O SUBROUTINES (IOPIC)

Subroutine	Function
OPPIC(LUN,FILE,NCOLS,NROWS,NBYTES,AGE)	Opens picture files
RDPIC(LUN,IROW1,IROW2,BUFFER)	Reads picture files
WRPIC(LUN,IROW1,IROW2,BUFFER)	Writes picture files
CLOPIC(LUN)	Closes picture files

Listings of the balance of the programs and subroutines are contained on the following pages.

```
      PROGRAM PROC12
C
C
C
C  PROGRAM TO PERFORM 1 DIMENSIONAL DIGITAL PROCESSING
C  OF COMPRESSED IMAGE DATA.
C
C
C     THIS PROGRAM PROCESSES A BYTE IOPIC DISC FILE OF
C       SIZE SPECIFIED BY THE USER.  IT FIRST GETS CERTAIN
C       HEADER INFORMATION FROM THE USER INCLUDING THE
C       PICTURE SIZE AND INPUT FILE NAME.  IT THEN FINDS
C       THE AVERAGE VALUE OF THE INPUT PICTURE AND PROMPTS
C       THE USER FOR VALUES OF DELTA AND BETA TO BE USED
C       BY THE 1 DIMENSIONAL DPCM COMPRESSOR WITH LEAKY
C       PREDICTOR.  THE COMPRESSION IS THEN PERFORMED AND
C       THE RESULTING COMPRESSED SIGNAL IS WRITTEN TO FILE
C       'COMP.TMP'.  IMAGE STATISTICS OF THE COMPRESSED
C       IMAGE ARE THEN COMPUTED AND PRINTED.  THE NAME OF
C       THE FILE CONTIANING THE 15 ELEMENT FILTER IS THEN
C       GOTTEN FROM THE USER AND THE FILE IS READ.
C       USEING THE COMPRESSED IMAGE AND THE FILTER, THE
C       VECTOR OF CORECCTION FACTORS IS THEN COMPUTED.
C       NEXT THE COMPRESSED PICTURE IS FILTERED.
C       FINALLY, THE FILTERED PICTURE IN FILE 'COMPF.TMP'
C       IS DECOMPRESSED USING THE VALUES OF DELTA, BETA,
C       AVERAGE, AND THE VECTOR OF CORRECTION FACTORS
C       PREVIOUSLY COMPUTED.  THE OUTPUT FILE NAME IS
C       SUPPLIED BY THE USER.  IMAGE STATISTICS FOR THE
C       OUTPUT FILE ARE COMPUTED AND WRITTEN ON FOR$PRINT.
C
C
C
C
CCCCCCCCC
C
C
C
C  RESTRICTIONS:
C
C  CURRENT DIMENSIONS LIMIT THE PICTURE SIZE TO 2K BY 2K PIXELS
C
C
C
C
CCCCCCCCC
C
C
C  DECLARE
C
C
      INTEGER*4       ISIZ(2)          !PICTURE SIZE
      LOGICAL*1       BUF1(2048)       !BYTE I/O BUFFER
      REAL*4          BUF2(2048),PDF(256),SCALE(2),FILTER(15)
      INTEGER*2       ICOL(2048)       !FOR CORRECTION VEC.
C
C
      CHARACTER*80    MESSAG           !FOR HEADERS
      CHARACTER*40    FIL1,FIL2,FIL3   !INPUT,OUTPUT AND FILTERFILES
      CHARACTER*40    PROMPT
C
C
      COMMON  /BUFFR/ BUF1,BUF2
```

```
C
C
      DATA    SCALE   /Ø.,255./       !MIN AND MAX OF DATA
C
C
CCCCCCCCC
C
C  FILL THE MESSAGE BUFFER AND PRINT IT IN THE HEADER
C
      TYPE 2
2     FORMAT(5X'TYPE MESSAGE: '///)
      ACCEPT 4,MESSAG
4     FORMAT(A4Ø)
C
C
      PRINT 6,MESSAG
6     FORMAT(/////2ØX,'***** DIGITAL PROCESSING
      1 OF COMPRESSED IMAGE DATA *****'/1ØX,A8Ø///)
C
C
CCCCCCCCC
C
C  GET THE INPUT FILE NAME FROM THE USER
C
            PROMPT   ' TYPE INPUT FILE NAME PLEASE:  '
      TYPE B,PROMPT
B     FORMAT(5X,A4Ø)
      ACCEPT 6Ø,FIL1
            PROMPT = 'SIZE OF PICTURE(ROWS,COLS):    '
      TYPE 8,PROMPT
      ACCEPT 9,ISIZ(1),ISIZ(2)
9     FORMAT(2I5)
C
C  AND FIND THE AVERAGE
C
      CALL HIST3(1,SCALE,PDF,FIL1,IERR,ISIZ)
            IF(IERR.NE.Ø)   CALL TYPERR(1,IERR)
      CALL ENTROP(PDF,Ø,IERR,H,AVE,A1,A2,A3,A4,A5,I1,I2)
            IF(IERR.NE.Ø)   CALL TYPERR(2,IERR)
C
C
CCCCCCCCC
C
C  GET COMPRESSION PARAMETERS FROM THE USER
C
            PROMPT = 'TYPE BETA:  '
      TYPE 1Ø,PROMPT
1Ø    FORMAT(5X,A2Ø,$)
      ACCEPT 2Ø,BETA
2Ø    FORMAT(F1Ø.4)
            PROMPT = 'TYPE DELTA:  '
      TYPE 1Ø,PROMPT
      ACCEPT 4Ø,IDELT
4Ø    FORMAT(I5)
C
C
      CALL DPCM1E(FIL1,'DBA1:COMP.TMP',ISIZ,IDELT,BETA,AVE,IERR)
            IF(IERR.NE.Ø)   CALL TYPERR(3,IERR)
C
C
      CALL HIST3(1,SCALE,PDF,'DBA1:COMP.TMP',IERR,ISIZ)
            IF(IERR.NE.Ø)   CALL TYPERR(4,IERR)
      CALL HISOUT(1,SCALE,PDF,1,'DBA1:COMP.TMP','NONE')
      CALL ENTROP(PDF,1,IERR,A1,A2,A3,A4,A5,A6,A7,I1,I2)
```

```
                IF(IERR.NE.Ø)   CALL TYPERR(5,IERR)
C
C
CCCCCC
C
C  GET FILE NAME FOR THE FILTER OPEN, READ AND CLOSE THE FILE
C
C
                PROMPT = '  FILE FOR FILTER:  '
        TYPE 1Ø,PROMPT
        ACCEPT 6Ø,FIL3
6Ø      FORMAT(A4Ø)
C
C
                OPEN(UNIT=1,FILE=FIL3,STATUS='OLD',FORM='UNFORMATTED')
                READ(1),(FILTER(I),I = 1,15)
                CLOSE(1)
C
C
CCCCCC
C
C  COMPUTE THE CORRECTION VECTOR
C
                CALL CORE1D(FIL1,FILTER,ICOL,ISIZ,IERR)
                        IF(IERR.NE.Ø)   CALL TYPERR(6,IERR)
C
C
CCCCCC
C
C  FILTER THE PICTURE
C
C
                CALL LINFLT('DBA1:COMP.TMP','DBA1:COMPF.TMP',
        1                       ISIZ,FILTER,128,IERR)
                        IF(IERR.NE.Ø)   CALL TYPERR(7,IERR)
C
C
C
CCCCCCC
C
C  GET THE OUTPUT FILE NAME FROM THE USER
C
                TYPE 7Ø
7Ø              FORMAT(5X,'OUTPUT FILE:  ',$)
                ACCEPT 6Ø,FIL2
C
C
CCCCCC
C
C  PERFORM THE INVERSE DPCM USEING VERSION G.
C
                CALL MCPD1G('DBA1:COMPF.TMP',FIL2,ISIZ,IDELT,BETA,AVE,ICOL,IERR)
                        IF(IERR.NE.Ø)   CALL TYPERR(8,IERR)
C
C
CCCCCC
C
C  COMPUTE AND PRINT THE IMAGE STATISTICS FOR THE OUTPUT
C  FILE, PRINT THE END MESSAGE AND EXIT.
C
C
CCCCCC
C
C
```

```
                CALL HIST3(1,SCALE,PDF,FIL2,IERR,ISIZ)
                        IF(IERR.NE.Ø)   CALL TYPERR(9,IERR)
                CALL HISOUT(1,SCALE,PDF,1,FIL2,'NONE')
                CALL ENTROP(PDF,1,IERR,A1,A2,A3,A4,A5,A6,A7,I1,I2)
                        IF(IERR.NE.Ø)   CALL TYPERR(1Ø,IERR)
C
C
        PRINT 8Ø,MESSAG
8Ø      FORMAT(2ØX,'***** END OF RUN *****'/1ØX,A8Ø)
C
C
                STOP
                END
```

```
      PROGRAM PROC13
C
C
C
C  PROGRAM TO PERFORM 1 DIMENSIONAL DIGITAL PROCESSING
C  OF COMPRESSED IMAGE DATA.
C
C
C     THIS PROGRAM PROCESSES A BYTE IOPIC DISC FILE OF
C       SIZE SPECIFIED BY THE USER.  IT FIRST GETS CERTAIN
C       HEADER INFORMATION FROM THE USER INCLUDING THE
C       PICTURE SIZE AND INPUT FILE NAME.  IT THEN FINDS
C       THE AVERAGE VALUE OF THE INPUT PICTURE AND PROMPTS
C       THE USER FOR VALUES OF DELTA AND BETA TO BE USED
C       BY THE 1 DIMENSIONAL DPCM COMPRESSOR WITH LEAKY
C       PREDICTOR.  THE COMPRESSION IS THEN PERFORMED AND
C       THE RESULTING COMPRESSED SIGNAL IS WRITTEN TO FILE
C       'DBA1:COMP.TMP'.  IMAGE STATISTICS OF THE COMPRESSED
C       IMAGE ARE THEN COMPUTED AND PRINTED.  THE NAME OF
C       THE FILE CONTIANING THE 15 ELEMENT FILTER IS THEN
C       GOTTEN FROM THE USER AND THE FILE IS READ.
C       NEXT THE PICTURE IS DECOMPRESSED USING THE VALUES
C       OF DELTA, BETA AND THE AVERAGE PREVIOUSLY COMPUTED.
C       THE DECOMPRESSED IMAGE IS STORED IN FILE 'DBA1:COMPD.TMP'.
C       FINALL, THE DECOMRESSED IMAGE IS FILTERED AND WRITTEN
C       TO A FILE SUPPLIED BY THE USER.  IMAGE STATISTICS ARE
C       COMPUTED AND WRITTEN ON FOR$PRINT.
C
C
C
C
CCCCCCCCC
C
C
C
C  RESTRICTIONS:
C
C  CURRENT DIMENSIONS LIMIT THE PICTURE SIZE TO 512 BY 512
C
C
C
C
CCCCCCCCC
C
C
C  DECLARE
C
C
      INTEGER*4        ISIZ(2)          !PICTURE SIZE
      LOGICAL*1        BUF1(2048)       !BYTE I/O BUFFER
      REAL*4           BUF2(2048),PDF(256),SCALE(2),FILTER(15)
      INTEGER*2        ICOL(2048)       !FOR CORRECTION VEC.
C
C
      CHARACTER*80     MESSAG           !FOR HEADERS
      CHARACTER*40     FIL1,FIL2,FIL3   !INPUT,OUTPUT AND FILTERFILES
      CHARACTER*40     PROMPT
C
C
      COMMON  /BUFFR/ BUF1,BUF2
C
C
      DATA    SCALE   /0.,255./        !MIN AND MAX OF DATA
```

```
C
C
CCCCCCCCC
C
C  FILL THE MESSAGE BUFFER AND PRINT IT IN THE HEADER
C
      TYPE 2
2     FORMAT(5X'TYPE MESSAGE: '///)
      ACCEPT 4,MESSAG
4     FORMAT(A4Ø)
C
C
      PRINT 6,MESSAG
6     FORMAT(/////2ØX.'***** DIGITAL PROCESSING
      1 OF COMPRESSED IMAGE DATA *****'/1ØX,A4Ø///)
C
C
CCCCCCCCC
C
C  GET THE INPUT FILE NAME FROM THE USER
C
            PROMPT = ' TYPE INPUT FILE NAME PLEASE:  '
      TYPE 8,PROMPT
8     FORMAT(5X,A4Ø)
      ACCEPT 6Ø,FIL1
            PROMPT = 'SIZE OF PICTURE(ROWS,COLS):    '
      TYPE 8,PROMPT
      ACCEPT 9,ISIZ(1),ISIZ(2)
9     FORMAT(2I5)
C
C  AND FIND THE AVERAGE
C
      CALL HIST3(1,SCALE,PDF,FIL1,IERR,ISIZ)
            IF(IERR.NE.Ø)  CALL TYPERR(1,IERR)
      CALL ENTROP(PDF,Ø,IERR,H,AVE,A1,A2,A3,A4,A5,I1,I2)
            IF(IERR.NE.Ø)  CALL TYPERR(2,IERR)
C
C
CCCCCCCCC
C
C  GET COMPRESSION PARAMETERS FROM THE USER
C
            PROMPT = 'TYPE BETA:  '
      TYPE 1Ø,PROMPT
1Ø    FORMAT(5X,A2Ø,$)
      ACCEPT 2Ø,BETA
2Ø    FORMAT(F1Ø.4)
            PROMPT = 'TYPE DELTA:  '
      TYPE 1Ø,PROMPT
      ACCEPT 4Ø,IDELT
4Ø    FORMAT(I5)
C
C
      CALL DPCMIE(FIL1,'DBA1:COMP.TMP',ISIZ,IDELT,BETA,AVE,IERR)
            IF(IERR.NE.Ø)  CALL TYPERR(3,IERR)
C
C
      CALL HIST3(1,SCALE,PDF,'DBA1:COMP.TMP',IERR,ISIZ)
            IF(IERR.NE.Ø)  CALL TYPERR(4,IERR)
      CALL HISOUT(1,SCALE,PDF,1,'DBA1:COMP.TMP','NONE')
      CALL ENTROP(PDF,1,IERR,A1,A2,A3,A4,A5,A6,A7,I1,I2)
            IF(IERR.NE.Ø)  CALL TYPERR(5,IERR)
C
C
```

```
CCCCCC
C
C  GET FILE NAME FOR THE FILTER OPEN, READ AND CLOSE THE FILE
C
C
                PROMPT = '  FILE FOR FILTER:  '
        TYPE 1Ø,PROMPT
        ACCEPT 6Ø,FIL3
6Ø      FORMAT(A4Ø)
C
C
                OPEN(UNIT=1,FILE=FIL3,STATUS='OLD',FORM='UNFORMATTED')
                READ(1),(FILTER(I),I = 1,15)
                CLOSE(1)
C
C
CCCCCC
C
C  SET THE CORRECTION VECTOR
C
                DO 62 J = 1,512
62                      ICOL(J) = Ø
C
C
CCCCCC
C
C
C  GET THE OUTPUT FILE NAME FROM THE USER
C
                TYPE 7Ø
7Ø              FORMAT(5X,'OUTPUT FILE:  ',$)
                ACCEPT 6Ø,FIL2
C
C
CCCCCC
C
C  PERFORM THE INVERSE DPCM USEING VERSION G.
C
        CALL MCPDIG('DBA1:COMP.TMP','DBA1:COMPD.TMP',
        1                       ISIZ,IDELT,BETA,AVE,ICOL,IERR)
                        IF(IERR.NE.Ø)   CALL TYPERR(8,IERR)
C
C
CCCCCC
C  FILTER THE PICTURE
C
C
                CALL LINFLT('DBA1:COMPD.TMP',FIL2,ISIZ,FILTER,Ø,IERR)
                        IF(IERR.NE.Ø)   CALL TYPERR(7,IERR)
C
C
C
CCCCCCCC
C  COMPUTE AND PRINT THE IMAGE STATISTICS FOR THE OUTPUT
C  FILE, PRINT THE END MESSAGE AND EXIT.
C
C
CCCCCC
C
C
                CALL HIST3(1,SCALE,PDF,FIL2,IERR,ISIZ)
                        IF(IERR.NE.Ø)   CALL TYPERR(9,IERR)
                CALL HISOUT(1,SCALE,PDF,1,FIL2,'NONE')
                CALL ENTROP(PDF,1,IERR,A1,A2,A3,A4,A5,A6,A7,I1,I2)
```

```
                        IF(IERR.NE.Ø)   CALL TYPERR(1Ø,IERR)
C
C
      PRINT 8Ø,MESSAG
8Ø    FORMAT(2ØX,'***** END OF RUN *****'/1Ø%,A8Ø)
C
C
            STOP
            END
```

```
      SUBROUTINE DPCM1E(FIL1,FIL2,ISIZ,DELTA,BETA,AMEAN,IERR)
C
C
C
C     ROUTINE TO PERFORM ONE DIMENSIONAL DPCM ON A BYTE IOPIC
C     FORMAT FILE.
C
C
C     INPUTS:
C             FIL1  CHARACTER ARRAY CONTAINING THE INPUT
C                   FILID
C
C             FIL2  CHARACTER ARRAY CONTAINING THE OUTPUT
C                   FILID
C
C             ISIZ(2)   INTEGER VECTOR CONTAINING THE NUMBER
C                       OF ROWS (ISIZ(1)) AND COLUMNS (ISIZ(2))
C                       IN THE INPUT FILES.
C
C             DELTA     INTEGER SCALER CONTAINING THE SIZE OF
C                       THE QUANTIZER STEP SIZE USED IN THE DPCM
C                       LINEAR QUANTIZER (DELTA .GE. 2)
C
C
C     OUTPUTS:
C             IERR      INTEGER*4 SCALER FOR INDICATION OF ERROR RETURN
C                             IERR=0    INDICATES NO ERRORS DETECTED
C                             IERR=1    INDICATES ILLEAGAL DELTA
C
C
C
C     FILES READ:
C             AN INPUT UNSIGNED BYTE IOPIC FILE SPECIFIED
C             BY FIL1.  ISIZ(1) BY ISIZ(2).
C
C
C     FILES CREATED:
C             AN OUTPUT UNSIGNED BYTE IOPIC FILE SPECIFIED
C             BY FIL2.  ISIZ(1) BY ISIZ(2).  THIS FILE, ALTHOUGH
C             UNSIGNED, IS SHIFTED BY 128 UNITS.  RAW QUANTIZER
C             OUTPUT IS WRITTEN TO THE FILE AFTER SHIFTING
C             ONLY (IE. NO AFTER SCALING IS PREFORMED, ALL
C             CODES MAY BE PRESENT).
C
C
C     FILES MODIFIED:
C             NO FILES ARE MODIFIED BY THIS SUBROUTINE.
C
C
C     LUN'S USED
C             LUN=1 FOR INPUT FILE
C             LUN=2 FOR OUTPUT FILE
C
C
C    ***************************************************************
C
C
C  DECLARE
       CHARACTER*(*) FIL1,FIL2
```

```
      LOGICAL*1  BUF(1),INTBYT,REABYT
      INTEGER ISIZ(2),DELTA,BYTINT
      REAL*4  BYTREA
      COMMON /BUFFR/BUF
C
C
C  INITIALIZE
C
      IERR=Ø                    !NO ERRORS SO FAR
      BMEAN = (1. -BETA) * AMEAN      !DC TERM IN PREDICTION
C
C  CHECK FOR INPUT ERRORS
      IF (DELTA .LT. 2) GO TO 1Ø1Ø    !ERROR EXIT #1
C
C  OPEN THE INPUT FILE ON LUN=1 AND THE OUTPUT ON LUN=2
C
      CALL OPPIC(1,FIL1,ISIZ(1),ISIZ(2),1,'OLD')
      CALL OPPIC(2,FIL2,ISIZ(1),ISIZ(2),1,'NEW')
C
C  LOOP OVER ROWS
      DO 5ØØ K=1,ISIZ(1)      !K IS THE ROW INDEX
C  TELL THE USER WHERE YOU ARE
      TYPE 2,K
2     FORMAT(1H+,3X,' ROW ',I5,3X,'IN DPCM1D )
C
      CALL RDPIC(1,K,K,BUF)   !READ THE K'TH ROW INTO THE BUFFER
C
C  FOR THE FIRST COLUMN, SET THE DELAY BUFFER TO ZERO
C
      ALAST=Ø.                          !ALAST IS THE DELAY BUFFER
C
C  NOW START THE DPCM LOOP FOR THIS ROW
C
      DO 3ØØ J=1,ISIZ(2)      !J IS THE COLUMN INDEX
C
C ***************************************************************
C
            IZ=BYTINT(BUF(J))               !INPUT
            AE=IZ-ALAST                     !ERROR
            IE = JNINT(AE/DELTA)            !QUANTIZED ERROR
            IIE=IE+128                      !SHIFT THE ZERO
            IF(IIE .LT. Ø)IIE=Ø             !CLIP IF LOW
            IF(IIE .GT. 255)IIE=255         !CLIP IF HIGH
            BUF(J)=INTBYT(IIE)              !REPLACE INTO THE I/O BUFFER
            IE=IE*DELTA                     !SCALE BACK FOR RECONSRUCTION
            ALAST=BETA*(ALAST+IE)+BMEAN     !ESTIMATE INTO DELAY BUFFER
C
C  DONE WITH DPCM FOR THIS PIXEL
3ØØ   CONTINUE                        !NEXT PIXEL IN ROW
C ***************************************************************
C
C
C  DONE WITH THE ROW, WRITE THE BUFFER
C
      CALL WRPIC(2,K,K,BUF)
C
C  DONE WITH THIS ROW
5ØØ   CONTINUE                        !NEXT ROW
C
C
C  DONE WITH THE FILES, CLEAN UP
C
      CALL CLOPIC(1)
      CALL CLOPIC(2)
```

```
C
C   CLEAN RETURN
C
          RETURN
C
C
C   ERROR RETURNS
C
1Ø1Ø      IERR=1                                  ILLEGAL DELTA
          RETURN
C
C
C
          END
```

```
      SUBROUTINE MCPD1G(FIL1,FIL2,ISIZ,DELTA,BETA,AMEAN,ILINE,IERR)
C
C
C
C
C     ROUTINE TO PERFORM ONE DIMENSIONAL INVERSE DPCM ON A BYTE
C     IOPIC FORMAT FILE CREATED BY DPCM1D
C
C
C
C     INPUTS:
C             FIL1   CHARACTER ARRAY CONTAINING THE INPUT
C                        FILID
C
C             FIL2   CHARACTER ARRAY CONTAINING THE OUTPUT
C                        FILID
C
C             ISIZ(2)    INTEGER*4 VECTOR CONTAINING THE NUMBER
C                        OF ROWS (ISIZ(1)) AND COLUMNS (ISIZ(2))
C                        IN THE INPUT FILES.
C
C             DELTA      INTEGER*4 SCALER CONTAINING THE SIZE OF
C                        THE QUANTIZER STEP SIZE USED IN THE DPCM
C                        LINEAR QUANTIZER (DELTA .GE. 2)
C
C             BETA       REAL*4 SCALER CONTAINING THE COEFICIENT
C                        FOR THE LEAKY PREDICTION INTEGRATOR
C
C             AMEAN      REAL*4 SCALER CONTAINING THE AVERAGE
C                        OF THE ORIGINAL INPUT SIGNAL FOR
C                        USE IN THE PREDICTOR
C
C             ILINE    INTEGER*2 VECTOR (512) CONTIANING THE
C                      FULL 16 BIT PRECISION (SIGNED NUMBER)
C                      FOR THE FIRST ELEMENT OF EACH ROW.
C
C
C     OUTPUTS:
C             IERR       INTEGER*4 SCALER FOR INDICATION OF ERROR RETURN
C                              IERR=0     INDICATES NO ERRORS DETECTED
C                              IERR=1     INDICATES ILLEAGAL DELTA
C
C
C
C
C     FILES READ:
C             AN INPUT FILE CRREATED BY DPCM1D.  THIS FILE, A BYTE
C             IOPIC FORMAT OF SIZE ISIZ(1) BY ISIZ(2) CONTAINS THE
C             UNSCALED DPCM ERROR SIGNAL THAT HAS BEEN SHIFTED
C             128 UNITS SO AS TO BE NON NEGATIVE.  ALL VALID
C             UNSIGNED BYTE CODES MAY APPEAR SINCE THE FILE HAS
C             BEEN CODED BY NOT SCALING.
C
C
C
C     FILES CREATED:
C             AN OUTPUT FILE IN BYTE IOPIC FORMAT ISIZ(1) BY
C             ISIZ(2) CONTAINING THE RECONSTRUCTED PICTURE.
C             FILID FROM FIL2.
C
C
```

```
C       FILES MODIFIED:
C               NO FILES ARE MODIFIED BY THIS SUBROUTINE.
C
C
C       LUN'S USED
C               LUN=1 FOR INPUT FILE
C               LUN=2 FOR OUTPUT FILE
C
C
C
C     ************************************************************************
C
C
C   DECLARE
        CHARACTER*(*)   FIL1,FIL2
        LOGICAL*1       BUF(1),INTBYT,REABYT
        INTEGER*4       ISIZ(2),DELTA,BYTINT
        REAL*4          BYTREA
        INTEGER*2       ILINE(2048)
        COMMON /BUFFR/BUF
C
C
C   INITIALIZE
C
        IERR=0                      !NO ERRORS SO FAR
        BMEAN = (1.-BETA)*AMEAN !DC TERM IN LEAKY PREDICTOR
C
C   CHECK FOR INPUT ERRORS                  !DON'T CHECK (COMENT)
C       IF (DELTA .LT. 2) GO TO 1010    !ERROR EXIT #1
C
C   OPEN THE INPUT FILE ON LUN=1 AND THE OUTPUT ON LUN=2
C
        CALL OPPIC(1,FIL1,ISIZ(1),ISIZ(2),1,'OLD')
        CALL OPPIC(2,FIL2,ISIZ(1),ISIZ(2),1,'NEW')
C
C   LOOP OVER ROWS
        DO 500 K=1,ISIZ(1)          !K IS THE ROW INDEX
C   TELL THE USER WHERE YOU ARE
        TYPE 1,K
1       FORMAT(1H+,3X,' ROW ',I5,3X,'IN MCPD1D')
C
        CALL RDPIC(1,K,K,BUF)   !READ THE K'TH ROW INTO THE BUFFER
C
C   FOR THE FIRST COLUMN, SET THE DELAY BUFFER TO ZERO
C
        ALAST=ILINE(K)              !ALAST IS THE DELAY BUFFER
C
C   NOW START THE RECONSTRUCTION LOOP FOR THIS ROW
C
        DO 300 J=1,ISIZ(2)          !J IS THE COLUMN INDEX
C   ************************************************************
        IE=BYTINT(BUF(J))           !INTEGERIZE THE INPUT BYTE
        IE=IE-128                   !SHIFT TO RESTORE MEAN
        IE=IE*DELTA                 !DECODE
        ALAST=IE+ALAST              !RECONSTRUCT
        ILAST1=ALAST
        IF(ILAST1.LT.0)ILAST1=0 !CLIP IF LOW
        IF(ILAST1.GT.255)ILAST1=255     !CLIP IF HIGH
        BUF(J)=INTBYT(ILAST1)   !REPLACE I/O BUFFER ELEMENT
        ALAST = BETA*ALAST + BMEAN      !PREDICT THE NEXT VALUE
C
C ************************************************************************
C
C
```

```
C  DONE WITH INVERSE DPCM FOR THIS PIXEL
300     CONTINUE                                !NEXT PIXEL IN ROW
C ***************************************************************
C
C
C  DONE WITH THE ROW, WRITE THE BUFFER
C
        CALL WRPIC(2,K,K,BUF)
C
C  DONE WITH THIS ROW
500     CONTINUE                                !NEXT ROW
C
C
C  DONE WITH THE FILES, CLEAN UP
C
        CALL CLOPIC(1)
        CALL CLOPIC(2)
C
C  CLEAN RETURN
C
        RETURN
C
C
C  ERROR RETURNS
C
1010    IERR=1                                  !ILLEGAL DELTA
        RETURN
C
C
C
        END
```

```
      SUBROUTINE LINFLT(FIL1,FIL2,ISIZ,AKER,IEDGE,IERR)
C
C
C     SUBROUTINE TO AFFECT A ONEDIMENSIONAL LINEAR
C     FILTER ON A ISIZ(1) BY ISIZ(2) BYTE IOPIC FORMAT
C     DISK FILE.
C
C
C
C     INPUTS
C             FIL1,FIL2       THE NAMES OF THE INPUT AND OUTPUT FILES.
C
C             ISIZ            THE NUMBER OF ROWS (ISIZ(1)) AND COLUMNS
C                             (ISIZ(2)) IN THE PICTURES.
C
C             AKER            VECTOR CONTAINING THE 15 ELEMENTS OF THE
C                             FILTER KERNAL.
C
C             IEDGE           VALUE TO WHICH THE OUTSIDE OF THE PICTURE
C                             IS BUFFERED TO.
C
C             IERR            N/C IN THIS VERSION
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
C  DECLARE
C
      CHARACTER*(*)   FIL1,FIL2
      LOGICAL*1       BUF(2048),      READBYT
      INTEGER*4       ISIZ(2)
      REAL*4          ABUF(2052),AKER(15),    BYTREA
C
C
      COMMON  /BUFFR/ BUF,ABUF
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  OPEN THE INPUT AND OUTPUT FILES ON LUN'S 1 AND 2
C
C
      CALL OPPIC(1,FIL1,ISIZ(2),ISIZ(1),1,'OLD')
      CALL OPPIC(2,FIL2,ISIZ(2),ISIZ(1),1,'NEW')
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  INIT THE FIRST AND LAST 7 ELEMENTS OF THE SCRATCH BUFFER
C
      DO 20 I = 1,7
         ABUF(I) = IEDGE
20    DO 30 I = ISIZ(2) + 8, ISIZ(2) + 14
         ABUF(I) = IEDGE
30
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  FILTER THE PICTURE
C
      DO 500 K = 1,ISIZ(1)                      !LOOP OVER ROWS
         CALL RDPIC(1,K,K,BUF)                  !READ THE K'TH ROW
         TYPE 80,K
```

```
8Ø                FORMAT(1H+.'   LINE ',I5,' IN LINFLT')
                  DO 1ØØ I = 1,ISIZ(2)             !LOOP ACROSS THE ROW
                     ABUF(I+7) = BYTREA(BUF(I))    !FILL THE SCRATCH BUFFER
1ØØ               CONTINUE
C
CCCCCCCCCCCCCC
C
C
                  DO 2ØØ J = 1,ISIZ(2)             !LOOP ACROSS THE ROW
                        JJ   J + 15                !DATA INDEX
                        SUM = Ø.                   !INIT
                        DO 15Ø I = 1.15            !LOOP ACROSS THE KERNAL
                              SUM = SUM + AKER(I)*ABUF(JJ-I)
15Ø                     CONTINUE
                        BUF(J) = REABYT(SUM)       !FILL THE OUTPUT BUFFER
2ØØ               CONTINUE                         !NEXT PIXEL IN ROW
                  CALL WRPIC(2,K,K,BUF)            !WRITE THE ROW
5ØØ         CONTINUE                               !NEXT ROW
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  DONE WITH FILTER, CLEAN UP
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
            CALL CLOPIC(1)
            CALL CLOPIC(2)
C
C
            RETURN
            END
```

```
      SUBROUTINE HIST3(ITYPE,SCALE,PDF,FIL,IERR,ISIZ)
C
C
C     ROUTINE TO COMPUTE THE PDF OF A PICTURE IN AN IOPIC
C     FILE.
C
C
C     INPUTS
C             TYPE    TYPE OF DATA IN FILE
C
C               TYPE=
C                     1   FOR UNSIGNED BYTE DATA
C                     3   FOR SIGNED BYTE DATA
C                     2   FOR INTEGER*2 DATA
C                     4   FOR REAL*4 DATA
C
C             FIL             CHARACTER ARRAY CONTAINING A
C                             FILID
C
C             ISIZ(2)         INTEGER*4 VECTOR CONTAINING THE
C                             NUMBER OF ROWS (ISIZ(1))AND
C                             COLUMNS (ISIZ(2)).
C
C             SCALE(2)        REAL*4 VECTOR CONTAINING THE MIN
C                             (SCALE(1)) AND MAX (SCALE(2)) VALUES
C                             EXPECTED FOR REAL*4 OR INTEGER*2 DATA
C                             TYPES.
C
C
C     OUTPUTS:
C
C             PDF(256)        REAL*4 ARRAY CONTAINING 256 PDF
C                             VALUES.
C
C             IERR            ERROR RETURN
C                     IERR=Ø FOR NO ERRORS
C
C
C ***************************************************************
C                       NOTES
C ***************************************************************
C
C     FOR UNSIGNED BYTE DATA, PDF(1) IS THE ESTIMATED PROBABILITY
C     THAT THE SIGNAL HAS A VALUE OF ZERO OR LESS.  PDF(256) IS
C     THE ESTIMATED PROBABILITY THAT THE SIGNAL IS 255 OR MORE.
C     OTHERWISE, THE PROBABILITY THAT THE SIGNAL TAKES ON THE
C     VALUE OF N IS ESTIMATED BY PDF(N+1).
C
C
C     FOR INTEGER*4 DATA THE SIGNAL IS QUANTIZED LINEARLY TO 256
C     LEVELS AFTER BEING SHIFTED.  THE AMOUNT OF THE SHIFT IS
C     SCALE(1).  THE STEP SIZE OF THE QUANTIZER IS:
```

```
C
C               (SCALE(2)-SCALE(1))/255.
C
C       THE QUANTIZER OUTPUT, O IS GIVEN BY:
C
C               1    I .LT. SCALE(1)
C               (I-SCALE(1))/DELTA +1    SCALE(1) .LT. I .LT. SCALE(2)
C               256  I .GT. SCALE(2)
C
C
C       FOR SIGNED BYTE INPUT THE ABOVE QUANTIZER IS USED WITH
C       SCALE(1)=-128 AND SCALE(2)=+127.
C
C
C       FOR REAL*4 DATA THE SAME QUANTIZER IS USED
C
C
C
C       THE INPUT FILE IS OPENED ON LUN=1
C
C
C ***********************************************************************
C
C
C
C
C  DECLARE
      LOGICAL * 1 BYBUF(4096),INTBYT,READYT
      CHARACTER*(*)  FIL
      REAL*4 REBUF(2048),SCALE(2),PDF(256),BYTREA
      INTEGER*2 IBUF(2048)
      INTEGER*4 ISIZ(2)
      INTEGER*4  BYTINT
C
C  EQUIVALENCE THE I/O BUFFERS SINCE ONLY ONE IS USED
C
      EQUIVALENCE (BYBUF,REBUF),(REBUF,IBUF)
C  AND PUT IT IN THE COMMON BLOCK
      COMMON /BUFFR/REBUF
C
C
C  CLEAR THE PDF ACCUMULATORS
C
      DO 10 K = 1,256
10            PDF(K) = 0.
C
C
C
C  OPEN THE FILE FOR READS
C
      IF(ITYPE .NE. 3) CALL OPPIC (1,FIL,ISIZ(2),ISIZ(1),ITYPE,'OLD')
      IF(ITYPE .EQ. 3) CALL OPPIC (1,FIL,ISIZ(2),ISIZ(1),1,'OLD')
C
C
C ***********************************************************************
C
C  STEERING
C
C
      IF(ITYPE .EQ. 1) GO TO 500      !UNSIGNED BYTE
C
C
      IF (ITYPE .EQ. 3) SCALE(1)=-128.        !SIGNED BYTE
      IF (ITYPE .EQ. 3) SCALE(2)= 127.        !SIGNED BYTE
```

```
C
C
C  FOR CONVERSIONS
C
      DELTA=(SCALE(2)-SCALE(1))/255.
C
C
C  CONTINUE STEERING
C
      IF(ITYPE .EQ. 3) GO TO 1000      !SIGNED BYTE
      IF(ITYPE .EQ. 2) GO TO 1500      !INTEGER
      IF(ITYPE .EQ. 4) GO TO 2000      !REAL
C
C
C
C  UNSIGNED BYTE DATA
C
500   CONTINUE
C
C
      DO 900 K=1, ISIZ(1)      !LOOP OVER ROWS
C  TELL THE USER WHERE YOU ARE
      TYPE 1,K
1     FORMAT(1H~,3X,' ROW',I5,3X,' IN HIST3  )
      CALL RDPIC(1,K,K,BYBUF) !READ THE K'TH ROW
      DO 800 I=1,ISIZ(2)       !LOOP ACROSS THE ROW
      J = BYTINT(BYBUF(I)) + 1          !CONVERT
      PDF(J) = PDF(J) +1.      !ACCUMULATE
800   CONTINUE                 !NEXT PIXEL
900   CONTINUE                 !NEXT ROW
      GO TO 2500               !DONE COUNTING, NOW NORMALIZE
C
C
C  SIGNED BYTE
C
1000  CONTINUE
      DO 1400 K=1, ISIZ(1)     !LOOP OVER ROWS
      TYPE 1,K
      CALL RDPIC(1,K,K,BYBUF) !READ THE K'TH ROW
      DO 1300 I=1, ISIZ(2)     !LOOP ACROSS THE ROW
      J= BYBUF(I) + 129        !CONVERT
      PDF(J) = PDF(J) + 1.
1300  CONTINUE                 !NEXT PIXEL
1400  CONTINUE                 !NEXT ROW
      GO TO 2500               !DONE COUNTING, NOW NORMALIZE
C
C
C  INTEGER INPUT
1500  CONTINUE
C
      DO 1900 K=1,ISIZ(1)      !LOOP OVER ROWS
      CALL RDPIC(1,K,K,IBUF)   !READ THE K'TH ROW
      DO 1800 I=1,ISIZ(2)      !LOOP ACROSS THE ROW
      J = (IBUF(I)-SCALE(1))/DELTA + 1.5      !QUANTIZE,ROUND,CONVERT
      IF(J .LT. 1) J=1         !CLIP IF LOW
      IF(J .GT. 256) J = 256   !CLIP IF HIGH
      PDF(J)=PDF(J)+1.         !ACCUMULATE
1800  CONTINUE                 !NEXT PIXEL
1900  CONTINUE                 !NEXT ROW
C
      GO TO 2500               !DONE COUNTING, NOW NORMALIZE
C
C
C  REAL DATA
```

```
C
2000     CONTINUE
         DO 2400 K = 1,ISIZ(1)    !LOOP OVER ROWS
         CALL RDPIC(1,K,K,REBUF) !READ THE K'TH ROW
         DO 2300 I = 1, ISIZ(2)  !LOOP ACROSS THE ROW
         J= (REBUF(I)-SCALE(1))/DELTA + 1.5       !QUANTIZE,ROUND,CONVERT
         IF( J .LT. 1) J=1        !CLIP IF LOW
         IF( J .GT. 256) J=256    !CLIP IF HIGH
         PDF(J)=PDF(J) + 1.
C
2300     CONTINUE                 !NEXT PIXEL
2400     CONTINUE                 !NEXT ROW
C
C
C
C  ***************************************************************
C
C
C
C  NORMALIZE
2500     CONTINUE
C
C
         A= ISIZ(1) * ISIZ(2)     !NUMBER OF SAMPLES
         A= 1./A
C
C
         DO 3000 K=1,256          !LOOP OVER THE HISTOGRAM
3000     PDF(K)=PDF(K)*A          !NORMALIZE
C
C
C
C
C  CLOSE THE INPUT FILE
C
         CALL CLOPIC(1)
C
         RETURN
         END
```

```
      SUBROUTINE HISOUT(ITYPE,SCALE,PDF,IOUT,FIL1,FIL2)
C
C
C  THIS SUBROUTINE OUTPUTS A REAL*4 ARRAY OF 256 VALUES
C  REPRESENTING A PROBABILITY DENSITY FUNCTION (PDF), OR
C  HISTOGRAM.
C  OUTPUT IS TO A PREVIOUSLY OPEN LISTING FILE AND/OR
C  TO A DATA FILE SPECIFIED WHEN THE SUBROUTINE IS
C  CALLED.
C
C
C  INPUTS
C
C
C
C       ITYPE   INTEGER*4        VARIABLE
C               ITYPE IS THE CODE FOR TYPE OF DATA
C               THAT WAS HISTOGRAMED.  PRINTED ON THE
C               HEADER OF THE LISTING.
C
C               ITYPE=
C                       1       UNSIGNED BYTE DATA
C                       3       SIGNED BYTE DATA
C                       2       INTEGER*2 DATA
C                       4       REAL*4 DATA
C
C
C       PDF     REAL*4           ARRAY(256)
C               PDF IS THE ARRAY THAT IS OUTPUT IN
C               TABLE FORM TO THE LISTING FILE, AND
C               AS A SINGLE UNFORMATTED RECORD TO THE
C               DATA FILE.
C
C
C       IOUT    INTEGER*4        VARIABLE
C
C               IOUT CONTAINS THE CODE FOR OUTPUT
C
C               IOUT=
C                       Ø       NO OUTPUT
C                       1       OUTPUT TO LISTING FILE ONLY
C                       2       OUTPUT TO DATA FILE ONLY
C                       3       OUTPUT TO BOTH LISTING AND DATA FILES
C
C
C       SCALE   REAL*4           ARRAY (2)
C
C               SCALE(1)  CONTAINS THE MINIMUM AND
C               SCALE(2)  CONTAINS THE MAXIMUM VALUES
C               EXPECTED BY THE HISTOGRAM ROUTINE.
C               THESE VALUES ARE USE TO COMPUTE THE
C               TABLE ENTRY VALUES ONLY.
C
C
C       FIL1    CHARACTER*       ARRAY(*)
C
C               FIL1 CONTAINS THE NAME OF THE FILE
C               THAT THE PDF REPRESENTS.
C
C       FIL2    CHARACTER*       ARRAY(*)
C
C               FIL2 CONTAINS THE NAME OF THE DATA
C               FILE CREATED FOR WRITING.
C
```

```
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  DECLARE
C
C
      CHARACTER*(*) FIL1,FIL2
      REAL*4  PDF(256),SCALE(2)
C
C
C
C  CHECK FOR NO OUTPUT
C
      IF(IOUT .EQ. Ø)  RETURN
C
C
C  CHECK FOR OUTPUT TO DATA FILE ONLY
C
      IF(IOUT .EQ. 2) GO TO 2ØØØ
C
C CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C   OUTPUT TO LISTING FILE
C
C  PRINT HEADER
C
      PRINT 5
      PRINT 6,FIL1
      IF (IOUT .EQ. 3)PRINT 7,FIL2     !PRINT OUTPUT FILE NAME
      PRINT 12
C
5     FORMAT(1H1////)
6     FORMAT(' ',15X,'  PDF OF DATA FROM FILE:  ',A4Ø)
7     FORMAT(' ',15X,'  WRITTEN TO FILE:        ',A4Ø)
C
C
C  COMPUTE DELTA
C
      DELTA=(SCALE(2)-SCALE(1))/255.
C
C  PRINT THE TABLE
C
      DO 1ØØØ I=1,25              !ROW IN TABLE
      N=(I-1)*1Ø+1                !INDEX OF THE FIRST ELEMENT IN ROW
      AENTRY = SCALE(1)+(N-1)*DELTA   !ENTRY VALUE OF FIRST ELEMENT
C
C
      PRINT 1Ø, AENTRY,(PDF(J),J=N,N+9)
1ØØØ  CONTINUE
C
C
      I = 26              !LAST LINE OF THE TABLE
      N = (I-1)*1Ø +1
      AENTRY = SCALE(1)+(N-1)*DELTA
      PRINT 11,AENTRY,(PDF(J),J=N,N+5)
      PRINT 12
1Ø    FORMAT(' ',15X,F8.2,1ØF1Ø.6)
11    FORMAT(' ',15X,F8.2,6F1Ø.6)
12    FORMAT(///15X,1Ø8('*')///)
C
```

```
C
C       IF( IOUT .NE. 3) GO TO 3ØØØ
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  DATA FILE
C
2ØØØ    CONTINUE
        OPEN(UNIT=1,FILE=FIL2,STATUS='NEW',FORM='UNFORMATTED')
        WRITE (1) PDF
        CLOSE(1)
C
C
3ØØØ    CONTINUE
C
C
C
        RETURN
        END
```

```
      SUBROUTINE ENTROP(PDF,IOUT,IERR,H,EEXI,EEXISQ,VAR,SD,PROMIN,
     1                  PROMAX,MINIM,MAXIM)
C
C
C
C     SUBROUTINE ENTROP
C
C     THIS ROUTINE CALCULATES THE ZERO MEMORY ENTROPY OF A
C     SIGNAL FROM ITS FIRST ORDER PROBABILITY DENSITY FUNCTION
C     (PDF) OR HISTOGRAM.
C
C
C     INPUT:
C
C             PDF(256) A REAL*4 ARRAY CONTAINING THE VALUES
C             OF THE PDF OF THE SIGNAL.
C
C
C             IOUT  INTEGER*4 VARIABLE DESCRIBING OUTPUT OPERATIONS
C             TO BE PERFORMED
C
C
C                     IOUT=Ø  NO OUTPUT
C                     IOUT=1  OUTPUT TO FORSPRINT ONLY
C                     IOUT=2  OUTPUT TO FORSTYPE ONLY
C                     IOUT=3  OUTPUT TO FORSPRINT AND FORSTYPE
C     OUTPUT:
C
C             H   A REAL*4 VARIABLE WHICH IS THE ZERO MEMORY
C             ENTROPY OF THE SIGNAL IN BITS PER SAMPLE.
C
C             EEXI, EEXISQ  THE REAL*4 VALUES OF THE EXPECTED
C             VALUES OF THE DATA AND THE EXPAECTED VALUE OF THE
C             SQUARE OF THE DATA.
C
C             VAR, SD THE REAL*4 VALUES OF THE VARIANCE AND
C             STANDARD DEVIATION OF THE DATA REPRESENTED BY
C             PDF.   NOTE UNSCALED AND UNSHIFTED BYTE DATA
C
C
C             MINIM,MAXIM  INTEGER*4 VARIABLES CONTAINING THE
C             MINIMUM AND MAXIMUM DATA VALUES FOR WHICH THE
C             CORESPONDING PDF'S ARE NONZERO.
C
C
C             PROMIN,PROMAX  REAL*4 VARIABLES CONTAINING THE
C             MONIMUM AND MAXIMUM PDF VALUES FOR ALL THE DATA.
C
C
C             IERR   EROR RETURN FOR NON PDF INPUT:
C                     IERR   Ø          NO ERROR DETECTED
C                     IERR = 1          ONE OR MORE PDF VALUES NEGATIVE
C                     IERR - 2          SUM OF PDF VALUES IS NOT 1
C                                       (WITHIN 1.E-Ø5)
C
C
C
C
C
C
C
C
C
```

```
C
C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C +++++++++++++++RESTRICTIONS++++++++++++++++++++++++++++++++++++++++++++++++
C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
C
C
C       PDF MUST BE DIVIDED INTO 256 CELLS.  (IE. BYTE INPUT SIGNAL)
C       AS A RESULT THE MAXIMUM ENTROPY COMPUTED WILL BE 8.
C
C
C
C ##############################################################################
C ##############################################################################
C ##############################################################################
C ##############################################################################
C
C
C
C  DECLARE
C
C
      DIMENSION PDF(256)
      REAL*8  EXI,  EXISQ
C
C
C
C  INITIALIZE
C
C
      H=Ø.              !ZERO OUT THE ENTROPY ACCUMULATOR
      SUM=Ø.            !ZERO OUT THE PDF ACCUMULATOR
      IERR=Ø            !INITIALIZE THE ERROR RETURN TO NULL
      EXI=Ø.            !ZERO THE ACCUMULATOR FOR THE MEAN
      EXISQ=Ø.          !AND THE ACCUMULATOR FOR THE MEAN SQUARE
      PROMIN = 1.       !SET THE MINIMUM POINTER TO THE MAX POSSIBLE
      PROMAX = Ø.       !SET THE MAX POINTER TO THE MIN POSSIBLE
C
C
C
C  LOOP OVER THE POSSIBLE SIGNAL VALUES
C
      DO 1ØØ I=1,256
      IF(PDF(I).LT.  Ø.) GO TO 1ØØØ    !PDF ERROR EXIT
      IF(PDF(I).EQ.  Ø.) GO TO 9Ø      !SPECIAL CASE
      EXI = EXI + PDF(I) * (I-1)
      EXISQ = EXISQ + PDF(I) * (I-1)**2       !ACCUMULATE MEAN SQUARE
      H=H + PDF(I) * ALOG(PDF(I))      !ACCUMULATE ENTROPY
      SUM = SUM + PDF(I)               !ACCUMULATE PDF
C
 9Ø   CONTINUE
1ØØ   CONTINUE
C
C
      VAR = EXISQ - EXI**2             !VARIANCE
      SD = SQRT(VAR)                   !STANDARD DEVIATION
      EEXI = EXI                       !SINGLE PRECISION FOR RETURN
      EEXISQ = EXISQ                   !SINGLE PRECISION FOR RETURN
C
C
C  FIND THE MINIMUM OF THE DATA BY SEARCHING FOR THE FIRST NON ZERO PDF
C
      DO 2ØØ I = 1,256                 !LOOP OVER DATA VALUES
```

```
                IF(PDF(I).NE.Ø.)GO TO 19Ø    !FOUND IT
        GO TO 2ØØ                            !LOOK SOME MORE
19Ø             MINIM = I - 1                !THIS I - 1S THE MINIMUM VALUE
                GO TO 21Ø                    !SO EXIT THE LOOP
2ØØ     CONTINUE                             !LOOP BACK
21Ø     CONTINUE                             !DONE

C
C  NOW FIND THE MAXIMUM OF THE DATA AS THE LAST VALUE FOR WHICH
C  THE PDF IS NON ZERO.  ALSO, FIND THE MINIMUM AND MAXIMUM
C  PDF VALUES.
C
C
        DO 3ØØ I=1,256                       !LOOP OVER DATA VALUES
                IF(PDF(I).NE.Ø) MAXIM= I-1       !UPDATE IF TRUE
C
                PRCMIN=AMIN1(PDF(I),PRCMIN)
                PRCMAX=AMAX1(PDF(I),PRCMAX)
C
C
3ØØ     CONTINUE
C
C
C
C  CHECK FOR PDF SUM ERROR
C
C
        IF(SUM .GT. 1.ØØØØ1 .OR. SUM .LT. .99990)IERR = 2
C
C
C
        H=-H/ALOG(2.)                        !UNITS ARE BITS
C
C
C  OUTPUT SECTION
C
C  CHECK FOR TYPE OF OUTPUT REQUIRED
C
                IF(IOUT .EQ. Ø) GO TO 4ØØ        !NO OUTPUT
                IF(IOUT .LT. 2) GO TO 35Ø        !PRINT ONLY
C
C
C  TYPE OUTPUT
C
                TYPE 32Ø
32Ø     FORMAT(///5X,' RESULTS FROM PDF ANALYSIS PERFORMED BY SUBROUTINE
     1ENTROP: ')
                TYPE 325,EEXI,SD,H
325     FORMAT(/5X,' MEAN: ',F9.4,' STANDARD DEVIATION: ',F9.4,' ZERO ME
     1MORY ENTROPY: ',F7.4)
                TYPE 33Ø,EEXISQ,VAR
33Ø     FORMAT(/5X,' MEAN SQUARE: ',E11.5,' VARIANCE: ',E11.5)
                TYPE 335,MINIM,MAXIM,PRCMIN,PRCMAX
335     FORMAT(/5X,' MINIMUM DATA: ',I5,' MAXIMUM DATA ',I5,' MINIMUM PR
     1OBABILITY: ',F9.6,' MAXIMUM PROBABILITY: ',F9.6)
C
C
C  CHECK FOR PRINTING OUTPUT
C
        IF(IOUT.NE.3)GO TO 4ØØ
C
35Ø     CONTINUE
C
C  PRINT THE OUTPUT
```

```
C
      PRINT 365
365   FORMAT(17X,'RESULTS FROM PDF ANALYSIS BY SUBROUTINE ENTROP'/)
      PRINT 37Ø
37Ø   FORMAT(25X,'MEAN',9X,'STANDARD DEVIATION',2X,
     1'ZERO MEMORY ENTROPY')
      PRINT 375, EEXI,SD,H
375   FORMAT(17X,5X,F1Ø.4,1ØX,F1Ø.4,1ØX,F1Ø.4/)
      PRINT 38Ø
38Ø   FORMAT(17X,5X,'MEAN SQUARE',12X,'VARIANCE')
      PRINT 385,EEXISQ,VAR
385   FORMAT(17X,5X,E1Ø.4,1ØX,E1Ø.4/)
      PRINT 39Ø
39Ø   FORMAT(21X,'MINIMUM DATA',8X,'MAXIMUM DATA')
      PRINT 395,MINIM,MAXIM
395   FORMAT(17X,7X,I6,14X,I6/)
      PRINT 397
397   FORMAT(18X,'MINIMUM PROBABILITY  MAXIMUM PROBABILITY')
      PRINT 4Ø5,PROMIN,PROMAX
4Ø5   FORMAT(22X,F1Ø.6,1ØX,F1Ø.6)
C END OF OUTPUT SECTION
C
4ØØ   CONTINUE
C
C  NORMAL RETURN
C
      RETURN
C
C
C ***********************************************************************
C                    ERROR RETURNS
C ***********************************************************************
C
C
C
C
C
C
1ØØØ  IERR=1
      RETURN
C
C
C
      END
```

```
      PROGRAM GAUS1
C
C
C                 PROGRAM TO GENERATE
C                 1 DIMENSIONAL GAUSIAN
C                 BLUR AND PSEUDO WEINER
C                 FILTERS.
C
C
C
C  DECLARE
C
      REAL*4  B(15),   DUMBUF(3Ø)       !TWICE 15
      COMPLEX*8        FFTBUF(15)
      CHARACTER*4Ø     FIL
C
C
      EQUIVALENCE      (FFTBUF(1),DUMBUF(1))
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  GET SIGMA, N/S FROM THE USER
C
      TYPE 1Ø
1Ø    FORMAT(1ØX,'***** GAUS1 *****'///5X,'TYPE SIGMA:   ',S)
      ACCEPT 2Ø,SIGMA
2Ø    FORMAT(F1Ø.4)
      TYPE 3Ø
3Ø    FORMAT(5X'TYPE N/S:   ',S)
      ACCEPT 2Ø,ANS
C
C
C
      TYPE 4Ø
      ACCEPT 5Ø,FIL
4Ø                FORMAT(5X,'FILE FOR BLUR FILTER.   ',S)
5Ø                FORMAT(A4Ø)
C
C
C  FORM THE BLUR FILTER
C
      SUM = Ø.
      DO 1ØØ J = 1,15
                  B(J) = EXP(-((J-8)**2)/SIGMA)
                  SUM = SUM + B(J)
1ØØ   CONTINUE
      DO 11Ø J = 1,15
                  B(J) = B(J)/SUM
11Ø   CONTINUE
C
C  WRITE THE RESULT
C
                  OPEN(UNIT=1,FILE=FIL,STATUS='NEW',FORM='UNFORMATTED')
                  WRITE(1)(B(J),J=1,15)
                  CLOSE(1)
C
C
C  FILL THE FFT BUFFER
C
```

```
      DO 115 J = 1,15
115             FFTBUF(MOD(J+7,15)+1) = CMPLX(B(J))
C
C
C  DO THE FORWARD TRANSFORM
C
C
                CALL FFTMIX(DUMBUF(1),DUMBUF(2),15,15,15,-2)
C
C
C  PRINT OUT THE RESULTS
C
      PRINT 200,SIGMA,ANS,FIL
200   FORMAT(20X,'BLUR FILTER FOR SIGMA = ',F10.4/
      1       20X,'                N/S   = ',F10.4/
      2       20X,'PLACED IN FILE:         ',A40///)
      PRINT 210
210   FORMAT(10X,'      J      ',6X,'REAL',6X,'IMAG',7X,'MOD',7X,'PSF')
      PRINT 220
220   FORMAT(10X,'     ***     ',6X,'****',6X,'****',7X,'***',7X,'***')
C
C
      DO 300 J = 1,15
              PRINT 250,J,FFTBUF(J),CABS(FFTBUF(J)),B(J)
250           FORMAT (10X,I10,4F10.4)
300   CONTINUE
C
C
C  NOW COMPUTE THE INVERSE FILTER
C
C
      DO 400 J = 1,15
              FFTBUF(J) = CONJG(FFTBUF(J))/(CABS(FFTBUF(J))**2+ANS)
400   CONTINUE
C
C
C  GET THE FILENAME FOR THE INVERSE FILTER
C
      TYPE 410
      ACCEPT 50,FIL
410   FORMAT(5X,'TYPE FILE FOR INVERSE FILTER:  ',S)
C
C
C  PRINT OUT THE INVERSE FILTER IN FREQUENCY
C
C
      PRINT 425,FIL
425   FORMAT(///20X,'INVERSE FILTER IN FILE:  ',A40/
      1          20X,'FREQUENCY REPRESENTATION:  '//
      1          14X,'J',9X,'REAL',6X,'IMAG'//)
C
C
      DO 500 J = 1,15
              PRINT 450,J,FFTBUF(J)
450   FORMAT(10X,I10,2F10.4)
500   CONTINUE
C
C
C  INVERSE TRANSFORM
C
      CALL FFTMIX(DUMBUF(1),DUMBUF(2),15,15,15,2)
C
C
```

```
C
C  SORT OUT THE INDEX AND TAKE THE REAL PART
C
      DO 55Ø J = 1,15
55Ø         B(MOD(J+6,15)+1)=REAL(FFT$JF(J))
      SUM = Ø.
C
C  NORMALIZE THE FILTER TO UNIT GAIN
C
      DO 6ØØ J = 1,15
6ØØ         SUM = SUM + B(J)
      DO 65Ø J = 1,15
65Ø         B(J) = B(J)/SUM
C
C  WRITE OUT THE RESULT
C
            OPEN(UNIT=1,FILE=FIL,STATUS='NEW',FORM='UNFORMATTED')
            WRITE(1)(B(J),J=1,15)
            CLOSE(1)
C
C
C  PRINT THE RESULTING INVERSE FILETER
C
      PRINT 7ØØ
7ØØ   FORMAT(//2ØX,'INVERSE FILTER IN SPACE DOMAIN:'//
      1     14X,'J',9X,'REAL'//)
      DO 75Ø J   1,15
            PRINT 725,J,B(J)
725         FORMAT(1ØX,I1Ø,F1Ø.4)
75Ø   CONTINUE
C
C
      STOP
      END
```

```
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C                CONVERT.FOR
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
C       THIS MODULE CONTAINS CONVERSION FUNCTIONS
C       FOR USING UNSIGNED BYTE DATA WITH VAX FORTRAN
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C       Ø7/2Ø/81         A. MASIA
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
C       FUNCTION         FROM             TO               TYPE
C       ********         ****             **               ****
C
C       INTBYT           INTEGER*4        LOGICAL*1        LOGICAL*1
C       BYTINT           LOGICAL*1        INTEGER*4        INTEGER*4
C       REABYT           REAL*4           LOGICAL*1        LOGICAL*1
C       BYTREA           LOGICAL*1        REAL*4           REAL*4
C
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
      LOGICAL FUNCTION INTBYT*1 (INT)
C
C
C     INTEGER*4 TO UNSIGNED BYTE CONVERSION
C
C  DECLARE
C
      INTEGER*4        INTEGER,INT
C
C
      INTEGER = INT
C  BRING INTO RANGE
C
      IF(INTEGER.GT. 255)      INTEGER=255
      IF(INTEGER .LT. Ø)       INTEGER=Ø
C
C
C   SUBTRACT 256 IF GREATER THAN HALF SCALE
C
      IF(INTEGER .GT. 127)     INTEGER = INTEGER-256
C
C
```

```
C  CONVERT
C
      INTBYT=INTEGER
C
C
      RETURN
      END
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      INTEGER FUNCTION BYTINT*4  (BYTE)
C
C
C              UNSIGNED BYTE TO INTEGER*4 CONVERSION
C
C  DECLARE
C
      LOGICAL*1       BYTE
C
C
      BYTINT = BYTE
C
C
      IF(BYTINT .LT. Ø)BYTINT = BYTINT+256    !CONVERT
C
C
      RETURN
      END
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      LOGICAL FUNCTION READBYT*1  (REAL)
C
C
C
C              REAL*4 TO UNSIGNED BYTE CONVERSION
C
C
      I = JNINT(REAL)
C
C
      IF(I .GT. 255)  I = 255
      IF(I .LT. Ø)    I = Ø
C
C
      IF(I .GT. 127)  I = I-256
C
C
      READBYT = I              !CONVERT
C
C
      RETURN
      END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
      REAL FUNCTION BYTREA*4  (BYTE)
C
C
C              UNSIGNED BYTE TO REAL*4 CONVERSION
C
C
C  DECLARE
      LOGICAL*1  BYTE
C
C
```

```
      I = BYTE          !INTEGERIZE
C
C
      IF(I .LT. Ø)I = I+256
C
C
      BYTREA = I        !CONVERT
C
C
      RETURN
      END
```

```
      SUBROUTINE LINFIL(FIL1,FIL2,ISIZ,AKER,IOPT,IEDGE,IERR)
C
C
C
C
C  ROUTINE TO PERFORM A CONVOLUTION OF A BYTE IOPIC FILE
C  (FIL1) OF DIMENSIONS ISIZ(1) ROWS BY ISIZ(2) COLUMNS
C  WITH A 15 BY 15 PIXEL REAL ARRAY, AKER.  OUTPUT
C  IS TO A BYTE IOPIC FILE (FIL2). EDGE TREATMENT IS
C  DETERMINED BY IOPT.  ERROR CODES ARE RETURNED FOR
C  INPUT ERRORS.
C
C
C
C
C
C  INPUTS
C       FIL1              A CHARACTER ARRAY CONTAINING THE
C                         INPUT FILEID.
C
C       FIL2     A CHARACTER ARRAY CONTAINING THE
C                         OUTPUT FILEID.
C
C       ISIZ(2)           INTEGER*4 ARRAY CONTAINING THE NUMBER
C                         OF ROWS, ISIZ(1) AND COLUMNS, ISIZ(2)
C                         IN THE INPUT AND OUTPUT FILES.
C
C       AKER(15,15)       A LOGICAL*4 ARRAY CONTAINING THE CONVOLUTION
C                         KERNAL TO BE APPLIED TO THE DATA.
C
C       IOPT              INTEGER*4 PARAMETER INDICATING
C                         WHICH ACTION TO TAKE AT THE EDGES.
C                IOPT=1   REGION OUTSIDE PICTURE ASSUMED ZERO.
C                         WITH NO ADJUSTMENT OF THE GAIN.
C                IOPT=2   REGION OUTSIDE THE PICTURE IS ASSUMED
C                         TO BE ZERO, BUT THE GAIN IS INCREASED
C                         SO THAT THE LOCAL DC REMAINS CONSTANT.
C                IOPT=3   ALL INDICIES ARE EVALUATED
C                         MODULO(THE PICTURE SIZE) SO THAT THE
C                         PICTURE 'WRAPS AROUND', IE. IS PERIODIC.
C                IOPT=4   ONLY THE CENTRAL ISIZ(1)-14 BY ISIZ(2)-14
C                         AREA OF THE PICTURE IS FILTERED.  THE 7
C                         COLUMNS AND ROWS ALONG EACH BOUNDRY ARE
C                         NOT FILTERED.
C
C
C       IEDGE             INTEGER*4 PARAMETER CONTAINING THE
C                         LOCAL ZERO FOR THE SIGNAL. USED
C                         WITH IOPT = 1 AND 2.
C
C
C
C  OUTPUTS
C
C
C       IERR              AN INTEGER*4 VARIABLE CONTAINING THE
C                         ERROR RETURN.
C                IERR=Ø   NO ERRORS DETECTED
C                IERR=1   NON UNIT GAIN OF FILTER
C                IERR=2   UNABLE TO OPEN INPUT FILE
C                IERR=3   UNABLE TO OPEN OUTPUT FILE
C
C
C
```

```
C
C  FILES READ
C
C       ONE FILE IS READ BY THIS SUBROUTINE
C               AN ISIZ(1) BY ISIZ(2) BYTE IOPIC FILE
C               NAMED ACCORDING TO DEC FILES-11 FILID
C               CONVENTIONS.  THE NAME OF THE FILE IS
C               CONTAINED IN THE VECTOR FIL1
C
C
C  FILES CREATED
C       ONE FILE IS CREATED BY THIS SUBROUTINE
C               AN ISIZ(1) BY ISIZ(2) BYTE IOPIC FILE
C               NAMED ACCORDING TO DEC FILES-11 FILID
C               CONVENTIONS.  THE NAME OF THE FILE IS
C               CONTAINED IN THE VECTOR FIL2.
C
C
C  FILES MODIFIED
C       NOFILES ARE MODIFIED BY THIS SUBROUTINE
C
C
C
C  SUBROUTINES REFERENCED
C
C       SUBROUTINE              MODULE
C       **********              ******
C
C
C       OPPIC                   IOPIC
C       RDPIC                   IOPIC
C       WRPIC                   IOPIC
C       INTBYT                  CONVERT
C       MOD                     FORTRAN INTRINSIC
C       WRAP                    WRAP
C       BYTINT                  CONVERT
C       BYTREA                  CONVERT
C
C
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
C
C
C
C
C
C  DECLARE
       CHARACTER*(*) FIL1,FIL2
       INTEGER*2 N(15),I(15)
       REAL*4 AKER(15,15),GAIN(15),BYTREA
       INTEGER*4 ISIZ(2)
       LOGICAL*1 BUFF(526,15),OBUFF(512)        !NOTE DIMENSIONS LIMIT
       LOGICAL*1 REABYT,INTBYT,EDGE
                                                !SIZE TO 512 WIDE PICS
C      INTEGER*4 BYTINT
C
C
       COMMON  /BUFFR/ BUFF
       COMMON  /BUFFR3/ OBUFF
```

```
C
C
      EDGE = INTBYT(IEDGE)              !LOCAL ZERO FOR IOPT = 1,2
C
C  INITIALIZE THE ERROR RETURN AND CHECK FOR INPUT ERRORS
C
C
      IERR=Ø
      SUM=Ø.
      DO 1Ø K = 1, 15                   !LOOP OVER THE FILTER ROWS
      DO 1Ø J = 1, 15                   !AND THE FILTER COLUMNS
1Ø          SUM=SUM+AKER(K,J)           !GAIN OF FILTER
      IF( SUM .LT. .9999 .OR. SUM .GT. 1.ØØØ1) IERR=1
      IF( IERR .NE. Ø )RETURN           !ERROR RETURN
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
C  OPEN THE INPUT FILE ON LUN=1 AND THE OUTPUT ON
C  LUN = 2.
C
C
C
      CALL OPPIC(1,FIL1,ISIZ(2),ISIZ(1),1,'OLD')
      CALL OPPIC(2,FIL2,ISIZ(2),ISIZ(1),1,'NEW')
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  CHECK FOR EDGE TREATMENT AND BRANCH
C
      IF(IOPT .EQ. 3) GO TO 1ØØ        !PERIODIC
C
      IF (IOPT .EQ. 4) GO TO 51ØØ
C
C
C  IOPT = 1 OR 2 SO ZERO OUT ALL THE BUFFERS
C
      DO 5Ø K=1,15              ! ALL 15 BUFFERS
      DO 49 J=1,ISIZ(2)+14      !LENGTH OF BUFFERS
            BUFF(J,K)=EDGE
49    CONTINUE
5Ø    CONTINUE
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  FOR ALL THREE OPTIONS
C
C  FILL THE NEXT 7 BUFFERS WITH THE FIRST 7 PICTURE ROWS
C
C
      DO 6Ø K=8,14
      KK=K-7                        !ROW TO READ
      CALL RDPIC(1,KK,KK,BUFF(8,K))  !READ THE ROW
6Ø    CONTINUE
C
```

```
C
C
C   IF IOPT .NE. 3 SKIP THE NEXT SECTION OF CODE
C
        GO TO 13Ø        !
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
C   FOR IOPT = 3
C
C   FILL THE FIRST 7 BUFFERS FROM THE END OF THE PICTURE
C
C
1ØØ     CONTINUE
C
C
C
        DO 12Ø K = 1,7
        KK = ISIZ(1) - 7 + K     !ROW TO READ-FROM PICTURE BOTTOM
             CALL RDPIC(1,KK,KK,BUFF(8,K))
12Ø     CONTINUE
C
C
C   GO BACK TO HANDLE NEXT 7 BUFFERS
C
        GO TO 5Ø                 !GO BACK
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
13Ø     CONTINUE
C
C
C
C   THE FIRST 14 BUFFERS ARE NOW FULL.  WRAP AROUND THE COLUMNS
C   FOR OPTION 3.
C
C
C
        IF (IOPT .NE. 3) GO TO 15Ø       !BRANCH IF NOT PERIODIC
        DO 14Ø K = 1,14
14Ø     CALL WRAP(K,ISIZ(2))             !DATA IS IN COMMON
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
15Ø     CONTINUE
C
C
C
C   THE FIRST 14 BUFFERS ARE NOW CORRECTLY FILLED.  START THE
C   FILTERING PROCESS INCLUDING NORMAL I/O FOR THE BULK OF THE
C   PICTURE.
C
C
C
C
C
C   LOOP OVER THE OUTPUT ROWS
C
```

```
      DO 1000 K = 1,ISIZ(1)-8          !DON'T DO THE LAST ROWS
C
C
      TYPE 160 , K                     !TELL THE USER WHERE YOU ARE
160   FORMAT(1H+,3X,' ROW ',I5,3X,'IN LINFIL')
C
C  NOW FIND THE REQUIRED BUFFER TO RECIEVE THE DATA, THE PICTURE
C  ROW TO READ FROM AND READ THE DATA.
C
C
      KK=K+7           !ROW TO READ
      KKK=K+13
      KKK=MOD(KKK,15)+1       !BUFFER TO RECIEVE THE DATA
            CALL RDPIC(1,KK,KK,BUFF(B,KKK))
C
C
C  FOR IOPT=3 WRAP AROUND THE FIRST AND LAST PARTS OF THE BUFFER
C
      IF(IOPT .EQ. 3) CALL WRAP(KKK,ISIZ(2))  !DATA IS IN COMMON
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
C  NOW COMPUTE N(L)=N(K,L), THE ROW INDEX FOR THE CONVOLUTION LOOP
C  FOR THIS VALUE OF K.
C
            DO 500 L = 1,15                !ALL POSSIBLE L'S
            N(L) = MOD(L+KKK-1,15)+1       !SORTS OUT THE INDEX
500         CONTINUE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
C  NOW LOOP ACROSS THE ROW OF THE OUTPUT BUFFER FOR THE SHIFT.
C
C
C
      DO 900 J = 1,ISIZ(2)
C
C
C  COMPUTE THE COLUMN INDEX OF THE INPUT BUFFER TO BE USED FOR THIS
C  SHIFT (J) AND KERNEL INDEX (M).  THE INDEX IS I(M).
C
            DO 600 M=1,15            !NUMBER OF ROWS IN KERNEL
            I(M) = J+M-1             !INPUT BUFFER COLUMN INDEX
600         CONTINUE
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
C  INITIALIZE THE FILTER OUTPUT TO ZERO
C
C
            SUM = 0.0
C
C
```

```
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  NOW LOOP OVER THE ROWS AND COLUMNS OF THE KERNEL AND COMPUTE
C  THE FILTER OUTPUT FOR THIS K AND J.
C
C
                DO 8ØØ L = 1,15          !ROW INDEX
                DO 7ØØ M = 1,15          !COLUMN INDEX
C
C
                        SUM = SUM + AKER(L,M)*BYTREA(BUFF(I(M),N(L)))
C
C
7ØØ             CONTINUE
8ØØ             CONTINUE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  FILTER IS NOW COMPLETE, CONVERT AND FILL THE OUTPUT BUFFER.
C
C
        IF(SUM .GT. 255.) SUM = 255.
        IF(SUM .LT. Ø.)   SUM = Ø.Ø
        OBUFF(J) = REABYT(SUM)           !CONVERT TO BYTE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  DONE WITH THIS ROW, GO BACK FOR THE NEXT ROW
C
9ØØ             CONTINUE         !END LOOP OVER J
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  THE ROW IS FINISHED,  WRITE IT OUT.
C
        CALL WRPIC(2,K,K,OBUFF)
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  GO BACK FOR THE NEXT ROW.
C
C
1ØØØ    CONTINUE         !END LOOP OVER K (ROW INDEX)
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  THE FIRST 8 ROWS AND THE BULK OF THE PICTURE ARE NOW FINISHED.
C
C  FILTER THE LAST 8 ROWS.
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
```

```
C  CHECK THE EDGE TREATMENT AND BRANCH.
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
      IF (IOPT .NE. 3) GO TO 2000      !BRANCH IF NOT PERIODIC
C
C
C  IOPT=3,  PERIODIC OUTPUT
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  LOOP OVER THE LAST ROWS OF THE OUTPUT PICTURE
C
C
      DO 1999 K= ISIZ(1) - 7,ISIZ(1)
      KK = K + 7
      KK = MOD(KK,ISIZ(1))+1  !ROW TO READ (FROM TOP OF PICTURE)
C
C
C  THE REST OF THE CODE IS NORMAL NOW
C
C
      KKK = K + 14
      KKK = MOD(KKK,15)+1     !BUFFER TO FILL
C
C
      CALL RDPIC(1,KK,KK,BUFF(8,KKK)) !READ THE ROW
C
C
C  SINCE IOPT = 3, SET THE FIRST AND LAST 7 ELEMENTS OF THE BUFFER
C
      CALL WRAP(KKK,ISIZ(2))          !DATA IS IN COMMON
C
C
C  NOW COMPUTE N(L) = N(L,K) FOR THIS K AS BEFORE
C
            DO 1500 L=1,15
            N(L)=MOD(L+KKK-1,15)+1  !SORTS OUT THE INDEX
1500        CONTINUE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  NOW LOOP ACROSS TH ROW WITH THE FILTER
C
C
            DO 1900 J=1, ISIZ(2)
C
C
C  COMPUTE THE COLUMN INDEX WITHIN THE BUFFER AS A FUNCTION OF
C  KERNEL INDEX (M) AND THIS SHIFT (J).
C
            DO 1600 M = 1,15
            I(M) = J+M-1            !INPUT BUFFER COLUMN INDEX
1600        CONTINUE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
```

```
C   INITIALIZE THE FILTER OUTPUT TO ZERO
C
C
                SUM = Ø.Ø
C
C
C
C   NOW LOOP OVER THE ROWS AND COLUMNS OF THE KERNAL AND COMPUTE THE
C   FILTER OUTPUT FOR THIS K,J.
C
C
                DO 18ØØ L = 1,15
                DO 18ØØ M - 1,15
                SUM= SUM + AKER(L,M)*BYTREA(BUFF(I(M),N(L)))
C
C
18ØØ            CONTINUE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C   FILTER IS NOW COMPLETE, CONVERT AND FILL THE OUTPUT BUFFER.
C
C
        IF(SUM .GT. 255.)SUM = 255.      !CLIP IF HIGH
        IF(SUM .LT. Ø.   )SUM = Ø.Ø      !CLIP IF LOW
C
C
                OBUFF(J)=REABYT(SUM)            !CONVERT TO BYTE
C
C
19ØØ            CONTINUE                 !NEXT SHIFT (END J LOOP)
C
C
C   DONE WITH THIS ROW,   WRITE IT.
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
        CALL WRPIC(2,K,K,OBUFF)
C
C
C   GO BACK AND GET THE NEXT ROW
C
1999    CONTINUE                         !NEXT ROW (END THE K LOOP)
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C   DONE WITH THE PICTURE FOR THIS OPTION.  THE NEXT SECTIONS OF CODE
C   ARE FOR THE OTHER OPTIONS,  SKIP THEM FOR COMMON HOUSEKEEPING AND
C   RETURN.
C
                GO TO 5555
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C   IOPT = 1 OR 2
C
```

```
2000            CONTINUE
C
C
C
C  LOOP OVER THE LAST 8 ROWS.
C
        DO 2999 K = ISIZ(1)-7,ISIZ(1)    !LAST ROWS OF OUTPUT
C
C
C
C  FIND WHICH BUFFER TO ZERO OUT
C
                KK = K + 7       !THIS IS THE ROW THAT WOULD HAVE BEEN READ
                KKK = K+14
                KKK = MOD(KKK,15)+1      !ZERO OUT THIS BUFFER
C
C  AND ZERO OUT THE BUFFER.
C
        DO 2010 J = 1, ISIZ(1)+14
2010            BUFF(J,KKK)=EDGE                    !ZERO THE BUFFER
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  THE REST OF THE CODE IS NORMAL.
C
C
C  COMPUTE N(L)= N(L,K) FOR THIS K.
C
C
                DO 2500 L = 1,15
2500            N(L)=MOD(L+KKK-1,15)+1   !SORTS OUT THE INDEX
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  NOW LOOP ACROSS THE ROW WITH THE FILTER.
C
C
                DO 2900 J=1,ISIZ(2)      !J IS THE FILTER SHIFT
C
C
C  COMPUTE THE COLUMN INDEX WITHIN THE BUFFER AS A FUNCTION OF KERNEL
C  INDEX, M FOR THIS SHIFT, J.
C
C
                DO 2600 M = 1,15         !ALL POSSIBLE M
                I(M)=J+M-1               !INDEX
2600            CONTINUE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  NOW INITIALIZE THE FILTER OUTPUT TO ZERO.
C
C
                SUM = 0.
C
C
C
C  LOOP OVER THE ROWS AND COLUMNS OF THE FILTER AND COMPUTE THE OUTPUT
C  FOR THIS ROW, K AND SHIFT, J.
```

```
C
C
        DO 28ØØ L=1,15
        DO 27ØØ M=1,15
C
C
                 SUM = SUM+AKER(L,M)*BYTREA(BUFF(I(M),N(L)))
C
C
27ØØ    CONTINUE
28ØØ    CONTINUE
C
C
C
C  FILTER IS COMPLETE FOR THIS PIXEL, CONVERT.
C
C
                 IF(SUM .GT. 255.)SUM = 255.
                 IF(SUM .LT. Ø.  )SUM = Ø.
C
C
                 OBUFF(J)= REABYT(SUM)
C
C
29ØØ    CONTINUE          !NEXT SHIFT (END OF LOOP OVER J)
C
C
C
C  DONE WITH THIS ROW,  WRITE IT.
C
C
                 CALL WRPIC(2,K,K,OBUFF)
C
C
C
C  DONE WITH THE ROW, GET THE NEXT ONE.
C
2999             CONTINUE
C
C
C
C  DONE WITH OPTIONS 1 AND 3.  BRANCH FOR COMMON HOUSEKEEPING AND RETURN.
C
C
                 IF(IOPT .NE. 2) GO TO 5555
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  IOPT = 2,  VARIABLE GAIN AT EDGES.
C
C
C
C  THIS EDGE TREATMENT IS INVOKED BY READING BACK THE OUTPUT DISK
C  FILE WRITTEN AS PART OF OPTION 1 AND APPLYING THE REQUIRED GAIN
C  FACTORS TO THE EDGES.  THE TOP/BOTTOM AND THE LEFT/RIGHT ARE
C  TREATED SEPERATLY.
C
C
C
C  FIRST SET UP THE GAIN FACTORS IN A REAL VECTOR.
C
C
        DO 41ØØ K= 1,8
```

```
                GAIN(K)= 1./(K+7)
                GAIN(K)=15.*GAIN(K)
41ØØ    CONTINUE
C
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  APPLY TO THE FIRST AND LAST 8 COLUMNS OF ALL THE ROWS.
C
C
C
                DO 45ØØ K= 1, ISIZ(1)
                CALL RDPIC(2,K,K,OBUFF) !READ IT INTO THE FIRST BUFFER
                DO 44ØØ J=1,8           !FIRST 8 COLUMNS
                SUM =GAIN(J)*BYTREA(OBUFF(J))
                IF(SUM .LT. Ø.) SUM = Ø.
                IF(SUM .GT. 255.)SUM = 255.
                OBUFF(J)=REABYT(SUM)
                JJ   1+ISIZ(2)-J        !END OF ROW
                SUM = GAIN(J)*BYTREA(OBUFF(JJ))
                IF(SUM .LT. Ø.) SUM = Ø.
                IF(SUM .GT. 255.)SUM = 255.
                OBUFF(JJ)=REABYT(SUM)
44ØØ    CONTINUE                !NEXT COLUMN (END LOOP ON J)
C
C
C  DONE WITH THE ROW, WRITE IT
        CALL WRPIC(2,K,K,OBUFF)
C
C
C
C  NOW GO GET THE NEXT ROW
C
45ØØ    CONTINUE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  NOW APPLY THE GAIN TERM THE THE FIRST 8 ROWS
C
C
        DO 49ØØ K=1,8   !LOOP OVER ROWS
                CALL RDPIC(2,K,K,OBUFF) !READ THE ROW INTO THE FIRST BUFFER
        DO 48ØØ J=1,ISIZ(2)             !LOOP ACROSS THE ROW
                SUM=GAIN(K)*BYTREA(OBUFF(J))
                IF(SUM .GT. 255.)SUM=255.
                IF(SUM .LT. Ø.  )SUM=Ø.
                OBUFF(J)=REABYT(SUM)    !CONVERT TO BYTE
48ØØ    CONTINUE
C  DONE WITH THE ROW, WRITE IT
                CALL WRPIC(2,K,K,OBUFF)
49ØØ    CONTINUE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C  NOW DO THE SAME TO THE LAST ROWS OF THE PICTURE
C
C
        DO 5ØØØ K=ISIZ(1)-7, ISIZ(1)    !LOOP OVER THE LAST ROWS
```

```
                CALL RDPIC(2,K,K,OBUFF) !GET THE ROW
                KK = K - ISIZ(1) + 8    !USE THE KK'TH GAIN TERM
                GA=GAIN(KK)
        DO 495Ø J=1,ISIZ(2)             !LOOP ACROSS THE ROW
                SUM = GA* BYTREA(OBUFF(J))
                IF(SUM .LT. Ø.) SUM=Ø.
                IF(SUM .GT. 255.)SUM = 255.
                OBUFF(J)=REABYT(SUM)    !CONVERT TO BYTE
495Ø    CONTINUE                        !NEXT PIXEL IN ROW
C
C  DONE WITH THE ROW, WRITE IT
C
                CALL WRPIC(2,K,K,OBUFF)
C
C  GET THE NEXT ROW
C
5ØØØ    CONTINUE
C
        GO TO 5555
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  CODE FOR OPTION 4
C
51ØØ    CONTINUE
C
C
        DO 511Ø K = 1,14        !ROW TO READ
        CALL RDPIC (1,K,K,BUFF(8,K))    !READ IT
511Ø    CONTINUE
C
C
C  PUT THE FIRST 7 ROWS IN THE OUPUT BUFFER AND WRITE THEM
C
        DO 512Ø K = 1,7
        DO 5115 J = 1,ISIZ(2)   !POINTS TO POSITION IN OUTPUT BUFFER
                OBUFF(J) = BUFF(JJ,K)
5115            CONTINUE
                CALL WRPIC(2,K,K,OBUFF) !WRITE THE K'TH ROW
512Ø    CONTINUE
C
C
C  DONE WITH THE FIRST 14 ROWS, FILTER THE BULK OF THE PICTURE.
C
C
        DO 53ØØ K = 8,ISIZ(1)-7         !LOOP OVER OUTPUT ROWS
C
C  FIND THE REQUIRED ROW TO READ THE BUFFER INTO, THE BUFFER
C  TO READ IT INTO, AND READ IT.
C
                KK = K + 7              !ROW TO READ
                KKK = K+13
                KKK = MOD(KKK,15)+1     !BUFFER TO ACCEPT IT
                CALL RDPIC(1,KK,KK,BUFF(8,KKK)) !READ THE ROW
C
C
C  COMPUTE N(L)=N(K,L), THE ROW INDEX IN THE CONVOLUTION LOOP
C  FOR THIS VALUE OF K.
C
C
        DO 52ØØ L = 1,15
```

```
                N(L) = MOD(L+KKK-1,15)+1
5200    CONTINUE
C
C
C
C  NOW LOOP ACROSS THE ROW IGNORING THE FIRST AND THE LAST 7 PIXELS.
C
        DO 5250 J = 8,ISIZ(2)-7          !ACROSS THE ROW
C
C
C  COMPUTE I(M)=I(M,J), THE COLUMN INDEX FOR THIS SHIFT, J.
C
        DO 5210 M = 1,15
5210            I(M) = J+M+1
C
C
C
C  ADDRESS TABLES ARE LOADED, BEGIN THE FILTERING FOR OUPUT PIXEL
C  K,J.
C
        SUM = 0.                 !INITIALIZE THE CONVOLUTION SUM
C
        DO 5230 L = 1,15
        DO 5220 M = 1,15
                SUM = SUM + AKER(L,M)*BYTREA(BUFF(I(M),N(L)))
5220    CONTINUE
5230    CONTINUE
C
C
C  FILTER IS COMPLETE SO FILL THE APPROPRIATE POSITION OF THE OUTPUT
C                       BUFFER
C
                IF (SUM .GT. 255.) SUM = 255.    !CLIP IF HIGH
                IF (SUM .LT. 0.   ) SUM = 0.     !CLIP IF LOW
                OBUFF(J)   REABYT(SUM)
C
C
5250    CONTINUE                 !END THE LOOP OVER J
C
C
C  NOW FILL IN THE FIRST AND THE LAST SEVEN PIXELS FROM THE INPUT
C                         BUFFER
C
                KKKK = MOD(KKK+7,15)+1  !BUFFER TO GET FROM
        DO 5260 J = 1,7
                OBUFF(J) = BUFF(J+7,KKKK)
                OBUFF(ISIZ(2)-7+J) = BUFF(ISIZ(2)+J,KKKK)
5260    CONTINUE
C
C  THE OUTPUT BUFFER IS NOW FULL, WRITE OUT THE K'TH LINE
C
C
                CALL WRPIC(2,K,K,OBUFF)
5300    CONTINUE                         !NEXT ROW (K)
C
C
C
C  NOW X'FER THE CONTENTS OF THE LAST 7 INPUT BUFFERS TO THE
C  OUTPUT WITHOUT FILTERING.
C
        DO 5400 K = ISIZ(1) - 6, ISIZ(1)         !LAST 7 ROWS
                KKK = K + 13
                KKK = MOD(KKK,15)+1
                KKKK = MOD(KKK + 7,15)+1         !POINTS TO INPUT BUFF
```

```
        DO 532Ø J = 1,ISIZ(2)                          !LOOP ACROSS THE OUTPUT
                OBUFF(J) = BUFF(J+7,KKKK)
5320    CONTINUE
C
C  WRITE OUT THE ROW
C
        CALL WRPIC(1,K,K,OBUFF)
5400    CONTINUE
C
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C  DONE WITH OPTION 4
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
5555            CONTINUE
C
C
C  DONE WITH THE PICTURE, COMMON HOUSEKEEPING AND RETURN FOR
C  IOPT= 1,2,3,4.
C
C
C
        CALL CLOPIC(1)
       -CALL-CLOPIC(2)-------
C
C
C
        RETURN
C
C
        END
```

10. APPENDIX 2

INSTRUCTIONS FOR SUBJECTIVE EVALUATIONS

The following was given to each judge as an instruction sheet before each set of subjective tests.

****

INSTRUCTIONS

You are being asked to compare the outputs of two different digital image processing systems. Both of these systems have been designed to increase the sharpness of images.

You will be shown six pairs of images. In each case the pictures have been processed by the two different systems. You must decide, for each pair, which of the two you prefer. If you prefer the picture on the left please place the pair under the card marked "L". If you prefer the picture on the right place it under the card marked "R".

In each case you must make a decision.

****

11. VITA

Mr. Masia was born in Jersey City, New Jersey and grew up in Evanston, Illinois and Cleveland Heights, Ohio. He was a full time student at the Rochester Institute of Technology between 1974 and 1980. While in residence at R.I.T. Mr. Masia was the first recipient of the Raymond Davis Scholarship awarded by the Society of Photographic Scientists and Engineers, was awarded a Fuji Photo Film Scholarship, and was named a R.I.T. Scholar. He was employed as a research assistant to Dr. Burt Carroll, as a teaching associate in the Photographic Science and Instrumentation Division and as a tutor of mathematics and statistics in the Learning Development Center.

Since leaving R.I.T. in 1980 Mr. Masia has been employed by the EIKONIX Corporation where he presently holds the position of Senior Scientist.